Higher Nationals - Summative Assignment Feedback Form

Student Name/ID	Fathima Sameeha Rimzan/ TJ54296		
Unit Title	Unit 01 – Programming		
Assignment Number	2	**Assessor**	Ms. Menisha Silva
Submission Date	30/10/2025	**Date Received 1st submission**	30/10/2025
Re-submission Date		**Date Received 2nd submission**	

Assessor Feedback:

Grade:	**Assessor Signature:**	**Date:**

Resubmission Feedback:

- Please note resubmission feedback is focussed only on the resubmitted work

Grade:	**Assessor Signature:**	**Date:**

Internal Verifier's Comments:

Signature & Date:

- Please note that grade decisions are provisional. They are only confirmed once internal and external moderation has taken place and grades decisions have been agreed at the assessment board.

Important Points:

1. It is strictly prohibited to use textboxes to add texts in the assignments, except for the compulsory information. eg: Figures, tables of comparison etc. Adding text boxes in the body except for the before mentioned compulsory information will result in rejection of your work.
2. Avoid using page borders in your assignment body.
3. Carefully check the hand in date and the instructions given in the assignment. Late submissions will not be accepted.
4. Ensure that you give yourself enough time to complete the assignment by the due date.
5. Excuses of any nature will not be accepted for failure to hand in the work on time.
6. You must take responsibility for managing your own time effectively.
7. If you are unable to hand in your assignment on time and have valid reasons such as illness, you may apply (in writing) for an extension.
8. Failure to achieve at least PASS criteria will result in a REFERRAL grade.
9. Non-submission of work without valid reasons will lead to an automatic RE FERRAL. You will then be asked to complete an alternative assignment.
10. If you use other people's work or ideas in your assignment, reference them properly using HARVARD referencing system to avoid plagiarism. You have to provide both in-text citation and a reference list.
11. If you are proven to be guilty of plagiarism or any academic misconduct, your grade could be reduced to A REFERRAL or at worst you could be expelled from the course
12. Use word processing application spell check and grammar check function to help editing your assignment.
13. Use **footer function in the word processor to insert Your Name, Subject, Assignment No, and Page Number on each pag**e. This is useful if individual sheets become detached for any reason.

STUDENT ASSESSMENT SUBMISSION AND DECLARATION

When submitting evidence for assessment, each student must sign a declaration confirming that the work is their own.

Student name: Fathima Sameeha Rimzan TJ54296	Assessor name: Ms. Menisha Silva	
Issue date: 24/08/2025	Submission date: 30/10/2025	Submitted on: 30/10/2025
Programme: Pearson BTEC HND in Computing		
Unit: **Unit 01 – Programming**		
Assignment number and title: 1- Training Fee Calculation System for KickBlast Judo.		

Plagiarism

Plagiarism is a particular form of cheating. Plagiarism must be avoided at all costs and students who break the rules, however innocently, may be penalised. It is your responsibility to ensure that you understand correct referencing practices. As a university level student, you are expected to use appropriate references throughout and keep carefully detailed notes of all your sources of materials for material you have used in your work, including any material downloaded from the Internet. Please consult the relevant unit lecturer or your course tutor if you need any further advice.

Guidelines for incorporating AI-generated content into assignments:

The use of AI-generated tools to enhance intellectual development is permitted; nevertheless, submitted work must be original. It is not acceptable to pass off AI-generated work as your own.

Student Declaration

Student declaration

I certify that the assignment submission is entirely my own work and I fully understand the consequences of plagiarism. I understand that making a false declaration is a form of malpractice.

Student signature: Sameeha7722@gmail.com Date: 24/08/2025

Unit 01: Programming

Assignment Brief

Student Name/ID Number	Fathima Sameeha Rimzan/ TJ54296
Unit Number and Title	**Unit 01 – Programming**

Academic Year	2024/2025
Unit Tutor	Ms. Menisha Silva
Assignment Title	**Training Fee Calculation System for KickBlast Judo.**
Issue Date	24/08/2025
Submission Date	30/10/2025

Submission Format

The assignment submission is in the form of the following.

- **An individual written report** written in a concise, formal technical style using single spacing and font size 12. Include all necessary screenshots of your coding and screenshots of the developed application as evidence. Referencing should be done using the Harvard Referencing system.
- All the source code files, Complete GUI System with the database.

Submit all above in a one single .zip file to the submission portal.

Unit Learning Outcomes

LO1. Define basic algorithms to carry out an operation and outline the process of programming an application.

LO2. Explain the characteristics of procedural, object-orientated and event-driven programming.

LO3. Implement basic algorithms in code using an IDE.

LO4. Determine the debugging process and explain the importance of a coding standard

Computing-related cognitive skills :

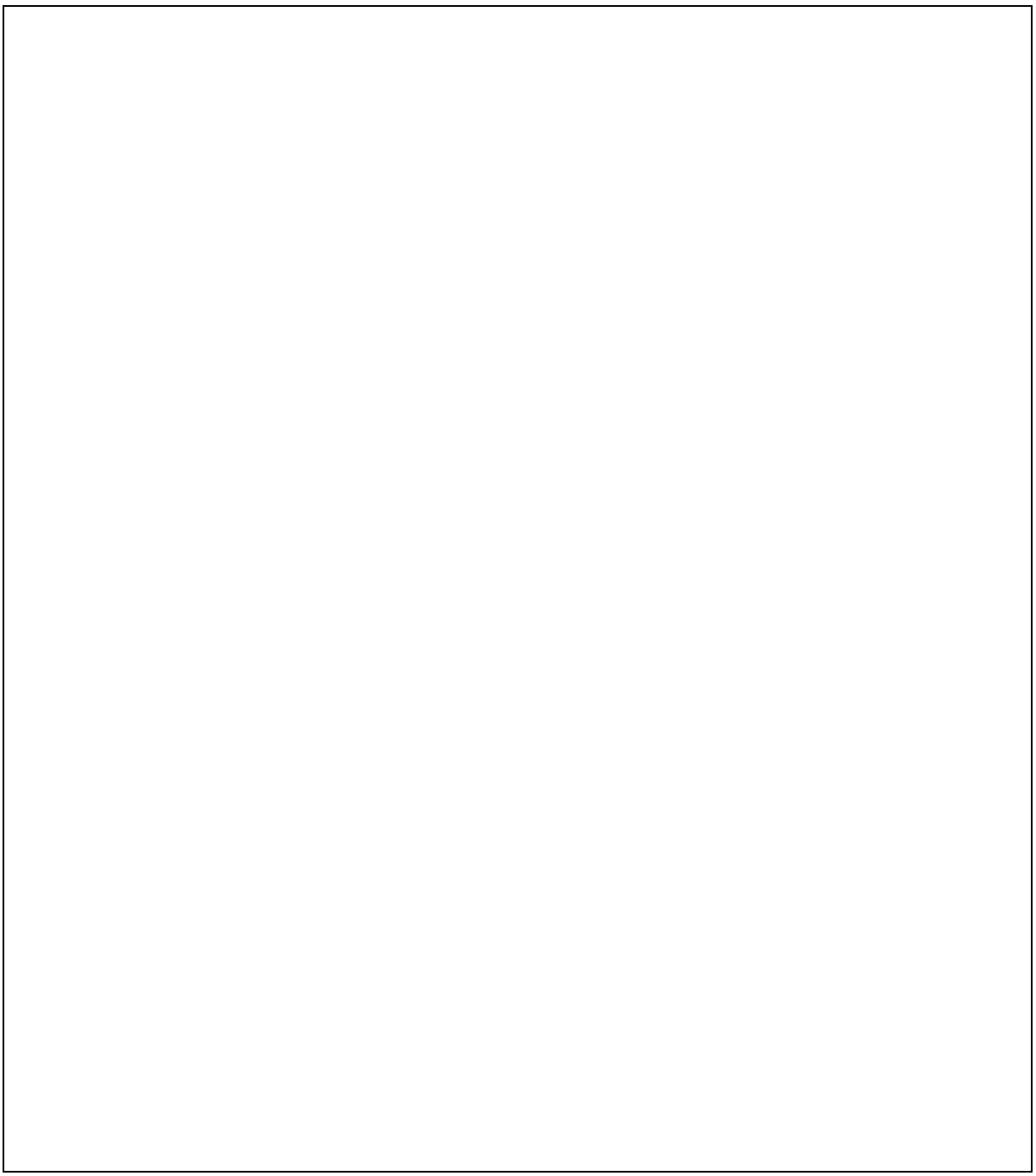
- **Demonstrate knowledge and understanding of essential facts, concepts, principles and theories relating to computing and computer applications**
- **Use such knowledge and understanding in the modelling and design of computer-based systems for the purposes of comprehension, communication, prediction and the understanding of trade-offs**
- **Recognise and analyse criteria and specifications appropriate to specific problems, and plan strategies for their solutions**
- **Critical evaluation and testing: analyse the extent to which a computer-based system meets the criteria defined for its current use and future development**
- **Methods and tools: deploy appropriate theory, practices and tools for the design, implementation and evaluation of computer-based systems.**

Computing-related practical skills :

- **The ability to specify, design and construct reliable, secure and usable computer-based systems**
- **The ability to evaluate systems in terms of quality attributes and possible trade-offs presented within the given problem**
- **The ability to deploy effectively the tools used for the construction and documentation of computer applications, with particular emphasis on understanding the whole process involved in the effective deployment of computers to solve practical problems**
- **The ability to critically evaluate and analyse complex problems, including those with incomplete information, and devise appropriate solutions, within the constraints of a budget.**

Generic skills for employability:

- **Intellectual skills: critical thinking; making a case; numeracy and literacy**
- **Self-management: self-awareness and reflection; goal setting and action planning**
- **Independence and adaptability; acting on initiative; innovation and creativity**
- **Contextual awareness, e.g. the ability to understand and meet the needs of individuals, business and the community, and to understand how workplaces and organisations are governed.**



Assignment Brief and Guidance:

Activity 1

A. The Fibonacci numbers are the numbers in the following integer sequence.
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ……..

In mathematical terms, the sequence Fn of Fibonacci numbers is defined by the recurrence relation.

$$F_n = F_{n-1} + F_{n-2}$$

B. Factorial of a non-negative integer, is multiplication of all integers smaller than or equal to n. For example, factorial of 6 is 6*5*4*3*2*1 which is 720.

$$n! = n * (n - 1) * \ldots\ldots 1$$

Define what an algorithm is and outline the characteristics of a good algorithm. Write the algorithms to display the Fibonacci series and the factorial value for a given number using Pseudo code. Determine the steps involved in the process of writing and executing a program and carry out an analysis of writing the code phase by discussing the potential challenges faced.

Take a sample number and dry run the above two algorithms. Show the outputs at the end of each iteration and the final output. Examine what Big-O notation is and explain its role in evaluating efficiencies of algorithms. Write the Python program code for the above two algorithms and critically evaluate their efficiencies using Big-O notation.

Activity 2

Compare and discuss what is meant by a Programming Paradigm and the main characteristics of Procedural, Object oriented and Event-driven paradigms and the relationships among them. Write small snippets of code as example for the above three

programming paradigms using a suitable programming language(s) and critically evaluate the code samples that you have given above in relation to their structure and the unique characteristics.

Activity 3 and Activity 4 are based on the following Scenario.

You have just started a new role as a Junior Software Developer at CUBE-GEN Software Solutions an independent software development company that designs and builds bespoke software solutions for various companies of different sizes that cover a range of different industries. The software that they design uses a wide range of technologies, from simple stand-alone programs to large web-based applications.

CUBE-GEN Software Solutions has been approached by a small, local company – KickBlast Judo – that specialises in providing judo training sessions to people from the local community. KickBlast Judo caters to people of all ages and experience, from expert to beginner.

KickBlast Judo has requested a simple program that will calculate the cost of training fees for their athletes each month.

The Chief Executive Officer (CEO) of the company has reviewed the client requirements and has determined that this is a suitable project for you to take on. The company wants to see how you use and apply the CUBE-GEN Software development environments and code standards.

The requirements are that KickBlast Judo wants a program that will allow a user to enter the following information:

- athlete name
- training plan
- current weight in kilograms (kg)
- competition weight category
- number of competitions entered this month
- option to add the number of hours private coaching.

For each athlete, the program should then output the following information:

- the athlete's name
- an itemised list of all costs for the month
- the total cost of training and competitions for the month
- how their current weight compares to their competition weight category.

KickBlast Judo currently have six athletes enrolled on their training program, but they would like the ability to register more.

You should use the additional information on the next page to help you when developing your

program.

The CEO has instructed you to develop a GUI application for the KickBlast Judo.

Once the program has been built, the CEO has asked you to report back to them on how you designed and developed the algorithms required, as well as how you converted these algorithms into a final program and to show any issues you encountered.

Additional information

Training Plan – Prices (Rs.)	
Beginner (2 sessions per week) – weekly fee	250.00
Intermediate (3 sessions per week) – weekly fee	300.00
Elite (5 sessions per week) – weekly fee	350.00
Private tuition – per hour	90.50
Competition entry fee – per competition	220.00

Weight categories	
Categories	Upper weight limit (kg)
Heavyweight	Unlimited (Over 100)
Light-Heavyweight	100
Middleweight	90
Light-Middleweight	81
Lightweight	73
Flyweight	66

- Athletes can receive a maximum of five hours' private coaching a week
- Only Intermediate and Elite athletes can enter competitions
- Competitions are held on the second Saturday of each month
- All prices and costs should be displayed as currency to two decimal places
- The program deals with user error by displaying suitable messages to the user and then prompting them for another go
- KickBlast Judo assumes that a month consists of four weeks.

Activity 3

Write the complete pseudocode to calculate the cost of training fees for KickBlast athletes for each month. Use the visual studio IDE (using C#.net) to implement the above algorithms. The developer can decide the methods which need to be included when developing the classes. Design the suitable database structure for keeping the data of the

above system.

Analyze the features of an Integrated Development Environment (IDE) and explain how those features help in application development. Evaluate the use of the Visual StudioIDE for your application development contrasted with not using an IDE.

Activity 4

4.1 Design and build a small GUI system for the above scenario and the application should be a complete functional system with all the functions which has described in the above scenario with the database structure which has designed in activity 3.

4.2 Examine debugging process and the features available in Visual studio IDE for debugging your code more easily. Evaluate how you used the debugging process to develop more secure, robust application with examples.

4.3 Explain and outline the coding standards you have used in your application development. Critically evaluate why a coding standard is necessary for the team as well as for the individual.

Reading:

Aho, A. V. et al. (1987) Data Structures and Algorithms. 1st Ed. Addison-Wesley. Hunt, A. et al. (2000) The Pragmatic Programmer: From Journeyman to Master. 1st Ed.

Addison-Wesley. McConnell, S. (2004) Code Complete: A Practical Handbook of Software Construction. 2nd Ed. Microsoft Press.

HN Global:

HN Global HN Global (2021) Reading Lists. Available at: https://hnglobal.highernationals.com/learning-zone/reading-lists

HN Global (2021) Student Resource Library. Available at: https://hnglobal.highernationals.com/subjects/resource-libraries

HN Global (2021) Textbooks. Available at: https://hnglobal.highernationals.com/textbooks

Pass	Merit	Distinction
LO1 Define basic algorithms to carry out an operation and outline the process of programming an application		
P1 Define an algorithm and outline the process in building an application. **P2** Determine the steps taken from writing code to execution.	**M1** Analyse the process of writing code, including the potential challenges faced.	**D1** Evaluate the implementation of an algorithm in a suitable language and the relationship between the written algorithm and the code variant.
LO2 Explain the characteristics of procedural, object-orientated and event-driven programming		
P3 Discuss what procedural, object-orientated and event-driven paradigms are; their characteristics and the relationship between them.	**M2** Compare the procedural, object-orientated and event-driven paradigms used in given source code of an application.	**D2** Critically evaluate the source code of an application that implements the procedural, object-orientated and event-driven paradigms, in terms of the code structure and characteristics.
LO3 Implement basic algorithms in code using an IDE		
P4 Write a program that implements an algorithm using an IDE.	**M3** Enhance the algorithm written, using the features of the IDE to manage the development process.	**D3** Evaluate the use of an IDE for development of applications contrasted with not using an IDE.
LO4 Determine the debugging process and explain the importance of a coding standard.		
P5 Explain the debugging process and the debugging facilities available in the IDE. **P6** Explain the coding standard you have used in your code.	**M4** Examine how the debugging process can be used to help develop more secure, robust applications.	**D4** Evaluate the role and purpose of a coding standard and why it is necessary in a team as well as for the individual.

Pearson Higher Nationals in Computing
Assignment 02: Programming

Fathima Sameeha Rimzan
24/08/2025

Acknowledgment

I would like to sincerely thank everyone who supported me throughout this journey. Firstly, a heartfelt thank you to my family for their love, patience, and encouragement to do this assignment, they've always been my biggest supporters and kept me going even when things got tough and hard at times

Additionally, I want to express my gratitude to Ms. Menisha Silva who is my lecturer for all of her help and advice with this assignment she was consistently friendly and approachable and her counsel truly improved my understanding of the situation and gave me a clear idea and her feedback helped my work and took me in the right direction whenever I felt uncertain or stuck. I value the time she invested in properly for explaining the situation and motivating us to keep trying our hardest and fullest to the maximum potential

Also, a big thank you to my friends and classmates who were always there to help, share ideas, and motivate me. Working together and learning from each other made this experience a lot more enjoyable. I truly appreciate everyone who played a part in helping me complete this assignment, thank you all for being a part of this journey as well.

Table of Content

Table of Figures

Table of Tables

Introduction

This programming assignment focuses on learning and applying different programming paradigms, concepts, and techniques such as procedural, object-oriented, and event-driven programming, and looks at how each one is used in real-world situations with examples. It also covers the implementation and development of a simple software solution using structured programming techniques and modern development tools.

Moreover, it involves working on a real-world scenario to design and develop a small application using C# and Visual Studio, this includes the design of the database tables, creation of pseudocodes, building a GUI and following proper coding standards.

The assignment further assesses the benefits of working with and without an Integrated Development Environment (IDE), examines debugging process and explains the features of an IDE. Overall, with these activities, the assignment aims to improve my understanding of logical problem solving, practical programming skills and the use of professional development environments.

Activity 01

1.1 Algorithm and Characteristics of a Good Algorithm

Definition of Algorithm

An algorithm is a step-by-step process used to solve a problem or perform a specific task (GeeksforGeeks, 2023). In simple terms, it is more like a recipe which tells you what to do exactly, in what order to get the final result. Algorithms take some input, follows a clear set of steps and produces an output.

Example- baking a cake

The process of baking a cake can be seen as an algorithm:

1. Gather all the ingredients needed such as flour, sugar, eggs and butter
2. Beat butter and sugar until it becomes light and creamy
3. Gradually add one egg at a time and beat the mixture
4. Then add in flour, and fold it with a spatula
5. Pour the batter into a baking pan.
6. Bake the batter at a 175 °C – 180 °C for a set amount of time.
7. let the cake cool before serving

This is considered as an algorithm because it starts with a clear aim of collecting the ingredients, follows defined steps like mixing, folding, baking and cooling and has results of a properly baked cake, therefore this is an algorithm as it showed the step-by-step process of baking a cake with inputs and outputs which will guarantees the same output every time when the instructions are followed in the right order.

An algorithm can be represented in many ways. Pseudocode is a way of writing the steps in simple, structured English that looks similar to programming but is very easy to read. This is an example of a pseudocode for baking the cake:

```
BEGIN
    Gather flour, sugar, eggs, butter
    Beat butter and sugar
```

Add eggs one at a time

Fold in flour

Pour into pan

Bake at 175°C – 180°C

Cool cake

OUTPUT "Cake is ready"

END

Likewise, flowchart shows the graphical representation of the same steps with shapes like ovals, rectangles and parallelograms. It also shows the flow of the algorithm visually and each shape is a call to different actions such as input, process or output.

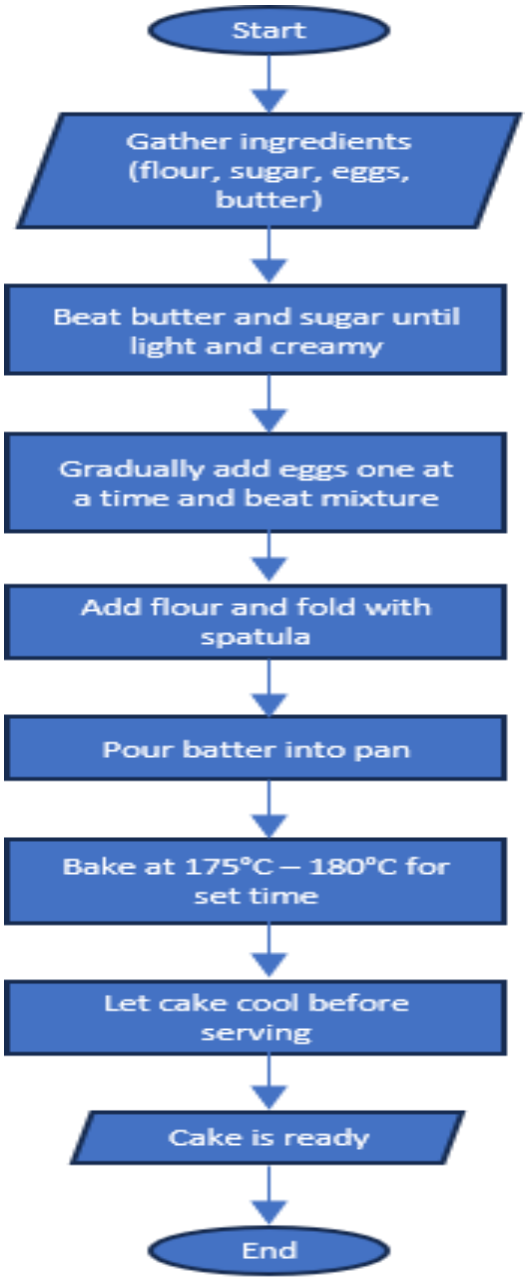


Figure 1: Cake baking flowchart (Author developed, 2025)

Characteristics of a Good Algorithm

An Algorithm shows the step-by-step process of solving a problem. However, not all algorithm is effective enough to be considered as a good algorithm, a good algorithm must have certain characteristics that make it reliable, efficient and practical to use. The characteristics below ensures that the algorithm works correctly and produces the right expected results.

1. Unambiguous

A good algorithm must be very clear and precise, leaving no room for confusion and each step should have only one meaning and be understood in the same way by everyone who reads or implements it. If the instructions are unclear or if the meaning is interpreted differently, the algorithm may produce wrong and unreliable results. For example, instead of saying "add eggs", a good algorithm would say "Gradually add one egg at a time and beat the mixture", this shows clarity which can be easy for the user to follow along with the steps and not miss the methods, and the algorithm would ensure consistency and correctness when is followed properly.

2. Input

Before an algorithm can begin, it should clearly state what inputs is needed, inputs are the raw data or value that the algorithm uses to generate and produce results. The algorithm cannot work without the right input. For example, in the baking a cake algorithm the inputs would be a list of ingredients such as flour, sugar, eggs and butter, these are used to later produce the results of a baked cake. A good algorithm specifies what inputs are needed and how much so the process can begin correctly.

3. Output

Every algorithm must produce at least one output, which is the result of the process. The output should be clearly defined for its purpose and must solve the problem that the algorithm was designed for. For example, the output of a cake recipe should be the results of a finished cake. Therefore, a good algorithm ensures that its outputs are correct, expected, and meaningful to the user.

4. Finiteness

A good algorithm should never take more than a certain number of steps to complete and it cannot run endlessly without producing a result. Finiteness guarantees the process is practical, realistic and applicable in real life. For example, if an algorithm for finding the largest number in a list keeps checking forever, it would be very useless. In conclusion, a finite algorithm guarantees that after a certain number of steps, it will stop and produce the result.

5. Feasibility

It is important for an algorithm to be realistic and practical to execute with the available sources like time, memory and processing power (TutorialsPoint, 2023). This means it should not require impractical or excessive volumes of data, or even steps that are impossible to perform. For instance, asking a computer to check every number up to infinity is not feasible and would take an infinity time, for a good algorithm to be implemented successfully, it must have a balance between practicality and efficiency.

6. Independent

A good algorithm should not be dependent on any particular technology or programming language, this means it can be described in simple, logical steps that anyone can understand, even without coding knowledge and experience. And later it can be implemented on any programming languages like Python, Java, C++ and so on. For example, the steps of baking a cake are the same, whether they are written in pseudocode, Sinhala, or English. Independence makes algorithm flexible and reusable.

In conclusion to that, a good algorithm is one that is clear, easy to understand and practical to implement. Its unambiguous steps, clear inputs and outputs, finite process, feasibility, and independence make it reliable and effective. By following these characteristics, algorithms can be effectively used in computer programming and real-world problem as well.

1.2 Fibonacci Series and Factorial

After understanding what an algorithm is and its characteristics, it is important to observe how algorithms can be used to solve real problems. Two common examples are generating the Fibonacci series and calculating the factorial of a number. Both problems can be solved using step-by-step logical instructions which can then be expressed in pseudocode.

Fibonacci Series Algorithm

The Fibonacci series is a sequence of numbers where each number is the sum of the two numbers before it. The series starts with 0 and 1, the first seven terms of the series are: 0, 1, 1, 2, 3, 5, 8. This sequence is generated using an algorithm that accepts a number n as input and produces the first n terms as output (Encyclopedia Britannica, 2024).

Pseudocode:

```
BEGIN
  INPUT n
  SET first = 0
  SET second = 1
  PRINT first, second
  FOR i = 3 TO n
    next = first + second
    PRINT next
    first = second
    second = next
  END FOR
END
```

Factorial Algorithm

The product of every positive integer that is less than or equal to a non-negative integer n is its factorial, for example, 5! = 5 × 4 × 3 × 2 × 1 = 120. A factorial algorithm takes the number n as input and calculates the result using repeated multiplication (TutorialsPoint, 2023).

Pseudocode:

```
BEGIN
  INPUT n
  SET factorial = 1
  FOR i = 1 TO n DO
    factorial = factorial * i
  END FOR
```

```
    PRINT factorial
END
```

1.3 Steps involved in Developing a Program

To make sure the software is accurate, effective, and simple to maintain, programmers use a systematic approach when creating and developing any program. This process is usually known as the Software Development Life Cycle (SDLC), SDLC is a method that helps the programmers identify the problem, develop solutions for that and create reliable programs.

Software Development Life Cycle (SDLC) can be simplified into stages such as:

1. **Defining the problem**

The most important step in program development is before directly writing any code or anything, the programmer must clearly understand what problem needs to be solved. For example, if the task is to create student grades, the programmer must know exactly what information needs to be entered like marks, and how the output should look like if in percentages or grade in letters. If the problem is not defined properly later when developing the program, the results may look incorrect.

2. **Planning the Solution**

After identifying the problem, the next step is to plan how to solve it, this is usually done by writing algorithms or drawing flowcharts to describe the steps in pictorial representation. This stage ensures that the logic is correct before coding the program and helps the programmer understand better.

3. **Coding the Program**

This step is known as the "coding" phase because in this the programmer translates the algorithm or the flowchart into a programming language such as Python, Java or C++. To produce a code that is easy to read and maintain, the programmer must follow syntax rules and structure the code.

4. Testing the Program

After coding, the program needs to be tested to make sure it works correctly, as testing helps to identify any errors such as syntax errors, logic errors or unexpected errors. The program may go through several rounds of debugging and fixes before it is ready.

5. Documenting the Program

This step makes sure that the program is easy to understand and maintain in the future, documentation includes writing instructions on how the program works, how to run it and explanations of the code with comments. A good documentation is very helpful for other programmers who may need to update, fix or improve the program later, so that they can understand the program without needing the original developer.

In summary, the process of developing a program follows the steps of the Software Development Life Cycle (SDLC), each step is important because it ensures that the final program is correct and easy to maintain, by following these steps the programmers can also reduce errors and find reliable solutions that meet the user's requirements.

1.4 Coding Phase

Explanation of Coding Phase

One of the most important stages in the development of a program is the coding stage, in this stage the algorithms and flowcharts created in the planning stage are converted into actual code using a programming language such as Python, Java or C++. This phase is where those ideas are turned into functional programs that a computer can execute, the program will function correctly, effectively and be able to be maintained in the future if the coding step is successful.

During the coding phase, my main aim was to translate the logic I designed into an actual working code, follow the correct syntax and structure of the programming language and make sure that my code was readable and clear. I also tested my code in small sections as I went along, which helped me catch errors early rather than waiting until the end.

Challenges I Faced in the Coding Phase and How I Overcame Them

1. Syntax Errors

Problem: One of the first issues I faced during my coding stage was the syntax errors, these usually happened when I forgot to close the brackets or punctuation in places like print statements or if conditions. For example, in Python, missing a closing parenthesis after a print statement immediately stopped my program from running.

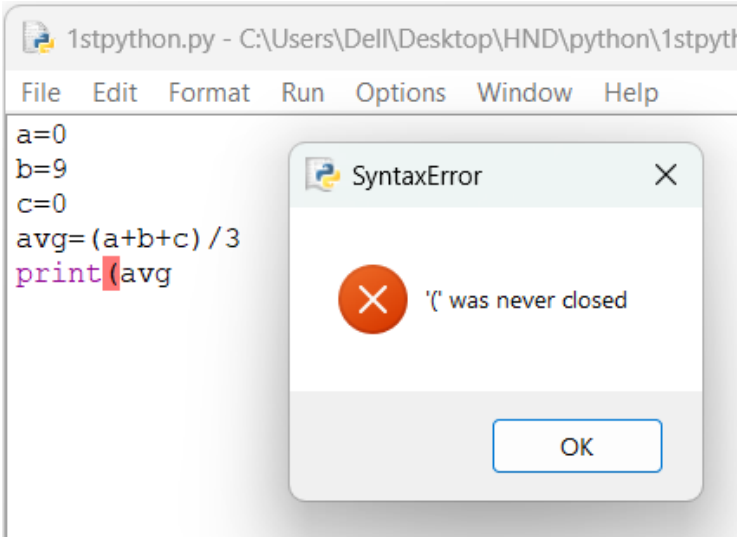


Figure 2: Syntax Error showing in Python IDLE (Author developed, 2025)

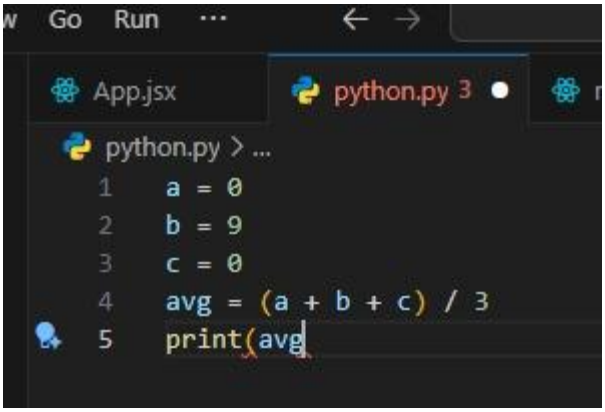


Figure 3: Syntax Error showing in Visual Studio Code (Author developed, 2025)

Solution: To overcome this, I later used Visual Studio Code, which underlined all my errors in real time, this allowed me to fix mistakes quickly as I typed, I also got into the habit of reviewing my code carefully and running small sections after writing them instead of writing everything at once.

2. Logic Errors

The biggest challenge I personally faced was logic errors, in this the program would run, but the output was not what I expected because my formula or structure was slightly wrong. For example, in one of my earlier attempts to calculate the average of three subjects I wrote:

```
# Program to find average of 3 subjects
math = 80
science = 70
english = 90
average = math + science + english / 3
print("Average Marks:", average)
```

Here, the problem was that only the english mark was divided by 3 because of the order of operations, so the average came out as 180 which is completely wrong. To fix this issue, I later carefully reviewed and used brackets to make sure the calculation happened in the right order:

```
average = (math + science + english) / 3
```

Now the average came out as 80, which matched my expected outcome, I also started testing my logic with different inputs and compared the results with manual calculations, this actually helped me quickly identify where my code went wrong.

3. Debugging and Handling Errors

Problem: during execution, I also encountered unexpected runtime errors, especially when inputs did not match what I expected. For instance, when dividing values, the program crashed if the divisor was zero.

Solution: To handle this, I added conditional checks before performing calculations to make sure invalid inputs would not break the program, I also learned to use try-except blocks in Python to catch and handle errors gracefully, so the program could continue running without crashing.

4. Understanding Complex Requirements

Another challenge I faced was understanding and converting more detailed requirements into actual code. Some parts of the task, like handling different conditions in calculations, initially felt very confusing.

Therefore, in order to solve this, I broke the program down into smaller sections and wrote pseudocode for each part, this made it easier to focus on one section at a time instead of trying to code everything at once. I also went back to the problem statement whenever I was unsure, to make sure that I was implementing the requirements correctly as I coded more.

For instance, when developing the login feature, I first focused only on the input validation part, instead of mixing it with the authentication logic, I wrote a simple pseudocode section just to handle checking whether the username and password fields were empty.

Pseudocode (Input Validation Section):

```
IF username is empty THEN
    DISPLAY "Please enter your username"
```

STOP

IF password is empty THEN
DISPLAY "Please enter your password"
STOP

Once this was working, I moved on to the authentication part separately

Pseudocode (Authentication Section):

CONNECT to database
EXECUTE query to check if a user exists with the entered username and password
IF a matching record is found THEN
LOGIN SUCCESSFUL
ELSE
DISPLAY "Invalid username or password"

By breaking the login process into smaller parts like this, I was able to focus on each section more clearly, avoid confusion and this step-by-step approach helped me implement the KickBlast Judo system easily.

5. Time management and efficiency

At certain times, the coding part took longer than I expected, especially when I got stuck with fixing errors, since I did not plan my time properly, even small issues delayed my progress.

To overcome this, I started setting clear mini goals for each coding session and focused on getting one part working perfectly before moving on to the next, this made the whole process more efficient and less overwhelming to me.

In conclusion, the coding phase was where my planned logic came to life and transformed into actual working code. Although, I faced many challenges such as syntax errors, logic mistakes, debugging issues and time management, I was still able to overcome them

through careful testing, using proper tools, reviewing my logic thoroughly and improving the quality of code as I progressed, by fixing and finding solutions I was able to build a working and reliable program that met the user requirements.

1.5 Dry Run of Algorithms

Dry Run of Fibonacci Series (n = 6)

Pseudocode:

```
SET n = 6
SET first = 0
SET second = 1
PRINT first, second
FOR i = 3 TO n
    next = first + second
    PRINT next
    first = second
    second = next
END FOR
```

Dry Run Table:

Iteration (i)	first	second	next	Expected Output	Actual Output
Start	0	1	-	0, 1	0, 1
3	0	1	1	0, 1, 1	0, 1, 1
4	1	1	2	0, 1, 1, 2	0, 1, 1, 2
5	1	2	3	0, 1, 1, 2, 3	0, 1, 1, 2, 3
6	2	3	5	0, 1, 1, 2, 3, 5	0, 1, 1, 2, 3, 5

Table 1: Dry Run Table of Fibonacci Series (Author Developed, 2025)

For an input of **n = 6**, the Fibonacci series was generated step by step and both the expected and actual outputs match at every iteration, this confirms that the algorithm and implementation are correct and the result is successful.

Dry Run of Factorial (n = 6)

Pseudocode:

```
SET n = 6
SET factorial = 1
FOR i = 1 TO n
    factorial = factorial * i
END FOR
PRINT factorial
```

Dry Run Table:

Iteration (i)	factorial Before	Operation	Expected factorial	Actual factorial
Start	1	-	1	1
1	1	1 × 1	1	1
2	1	1 × 2	2	2
3	2	2 × 3	6	6
4	6	6 × 4	24	24
5	24	24 × 5	120	120
6	120	120 × 6	720	720

Table 2: Dry Run Table of Factorial (Author Developed, 2025)

For the input **n = 6**, the factorial calculation was traced through each iteration, the expected and actual factorial values match at every step, resulting in a final correct output of 720, this shows that the algorithm works and the dry run is accurate.

1.6 Big-O Notation

Explanation of Big-O Notation

Big-O notation is a way to show how fast a program or algorithm works as the amount of data gets bigger (Programiz, 2024), it does not indicate the exact time but shows how the number of steps increases as the input increases. For example, if a program checks each item at once, its O(n). If it has a loop inside another loop its $O(n^2)$, which requires more time with large data sets. Big-O is like a speedometer for the code, allowing you to determine whether it will remain fast as the data increases in size. It is also helpful for comparing different programs or methods, allowing programmers to select the one that works best and does not slow down with more data.

Importance and Role of Big-O Notation

Big-O notation is important because it allows us to measure and understand the efficiency of algorithms regardless of the computer or programming language being used. When programmers create solutions, there can be many ways to write an algorithm and while all of them may produce the correct output, some may take more time or use more memory than others.

Big-O helps us identify these differences by providing a standard way to describe how the algorithm's performance changes as the input size increases, this makes it easier to compare different methods and select the one that is most practical especially when working with large amounts of data.

The role of Big-O notation provides guidance for optimization and decision making as well, it helps programmers predict whether an algorithm that works well for small problems will still remain efficient when applied to much larger ones. For instance, an algorithm that runs in O(n) time will usually handle growth in data more effectively than one with $O(n^2)$, which slows down much more quickly as inputs increase. Big-O notation ensures that the software we design is reliable and able to perform in real world conditions by focusing on "growth rate". In summary, Big-O plays a key role for improving code quality, as it demonstrates not only that an algorithm functions but also how effectively it will perform as challenges increase.

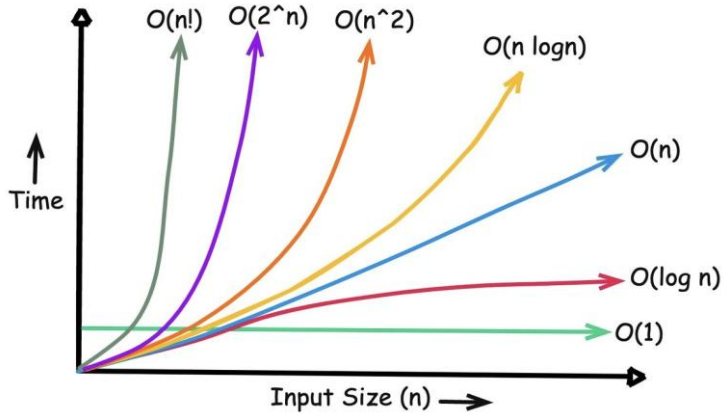


Figure 4: The Big O notation and its significance (Tarry Singh, 2024)

1.7 Conversion of Pseudocode to Python Programs

Fibonacci Series in Python

fibonacci.py - C:\Users\Dell\Desktop\HND\programming a

File Edit Format Run Options Window Help

```
# Fibonacci Series Program
n = 6
first = 0
second = 1

print("Fibonacci Series for n =", n)
print(first, second, end=" ")

for i in range(3, n + 1):
    next_num = first + second
    print(next_num, end=" ")
    first = second
    second = next_num
```

Figure 5: Python code for Fibonacci series typed in IDLE (Author developed, 2025)

```
== RESTART: C:\Users\Dell\Desktop\HN
Fibonacci Series for n = 6
0 1 1 2 3 5
>>>
```

Figure 6: Output of Fibonacci series program in Python Shell (Author developed, 2025)

Factorial in Python

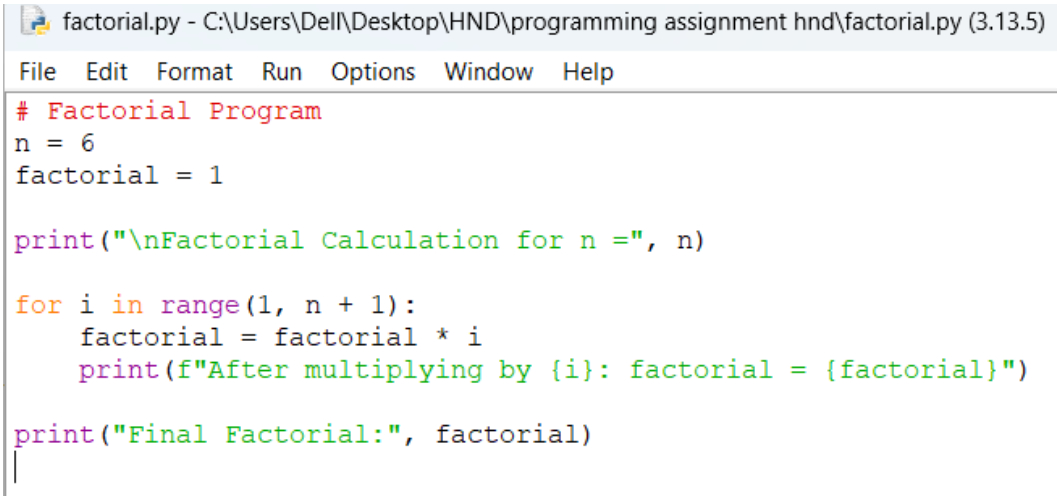


Figure 7: Python code for factorial program typed in IDLE (Author developed, 2025)

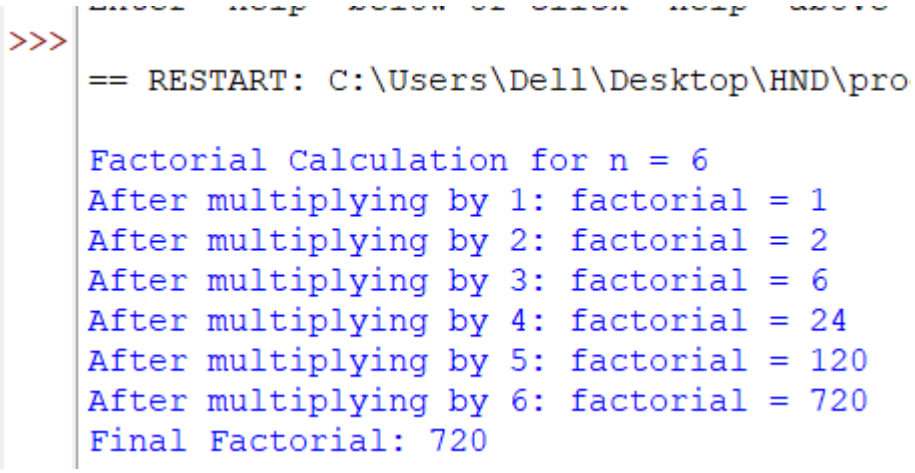


Figure 8: Output of factorial program in Python Shell (Author developed, 2025)

Evaluation of efficiency using Big-O Notation

The efficiency of an algorithm can be measured by its execution time or the number of steps as the input size increases and Big-O notation helps to express this in a standard way

For the Fibonacci series the algorithm we used repetitively calculates each term by adding the two previous terms, since it only needs a single loop that runs from 3 to n, the number of steps grow with the input size n. Therefore, the iterative Fibonacci program has a time complexity of O(n), this means when we double the number of terms we want to calculate, the execution time increases about twice as much, which is efficient and practical for large inputs.

In the same way for the factorial program, the algorithm multiples all the integers from 1 to n using a single loop. Like the Fibonacci program, the number of operations grows in direct proportion to n, the factorial program also has a time complexity of O(n). Both programs are very efficient as they avoid any unnecessary repeated calculations unlike recursive solutions that could need additional memory and more operations for the same result.

Additionally, both algorithms of Fibonacci and Factorial have a time complexity of O(n) because they use a single loop that increases with the input size. However, their space complexities are not the same, because it stores only three variables such as first, second and next so it uses O(1) space. The factorial algorithm also uses just one variable which is factorial, it also has O(1) space. This shows that they are not only fast but memory efficient. If we had used recursive versions, Fibonacci would become $O(2^n)$ due to repeated calculations and additional memory is needed for functional calls and factorial would be O(n) but with more space usage because of the recursion stack. This indicates why iterative solutions are more efficient in real world programming as they handle large inputs way better.

Although both iterative have O(n) time complexity, Fibonacci needs maintaining two variables and repeated additions, while Factorial needs only one multiplication loop, because multiplication is only done once per iteration, Factorial is computationally simpler even though their time efficiency is comparable.

In conclusion both algorithms run in O(n) time and O(1) space which makes them efficient and practical for real world programming tasks that need both speed and low memory usage.

1.8 Relationship between the written Algorithm and Python Code

1. Fibonacci Program

Written Algorithm:

```
BEGIN
  INPUT n
  SET first = 0
```

```
SET second = 1
PRINT first, second
FOR i = 3 TO n
    next = first + second
    PRINT next
    first = second
    second = next
END FOR
END
```

Python Implementation:

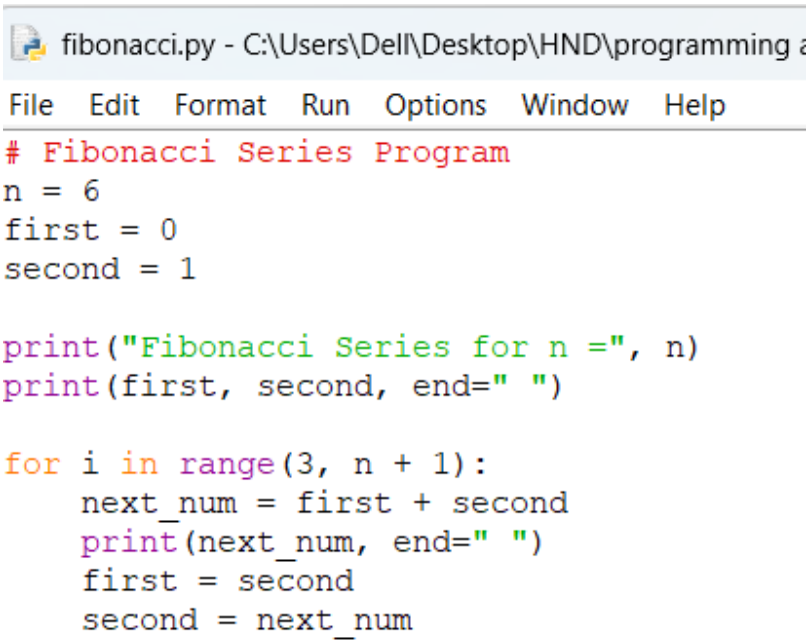


Figure 9: Python implementation of the Fibonacci algorithm for N = 6 (Author developed, 2025)

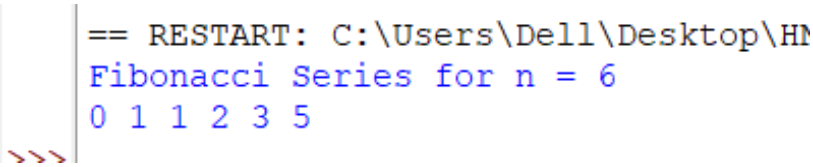


Figure 10: Output of Fibonacci series program in Python Shell (Author developed, 2025)

The pseudocode and the Python implementation of the Fibonacci program are directly related, the pseudocode first initializes the first two terms, then uses a loop to calculate the next term, prints it and updates the previous two values.

The same structure is also used in the Python code by storing the first two terms in variables n1 and n2, iterating from 3 to N using a loop, and executing each term using a print() statement. Both the pseudocode and the Python implementation follow the same logic which includes initialization, looping, calculation, and updating.

For example, in the pseudocode the line "SET next = n1 + n2" exactly matches to the Python line next = n1 + n2, likewise in the loop structure "FOR i = 3 TO N" matches the for i in range(3, n+1): loop in Python. This shows a clear one-to-one mapping between planning and implementation where the algorithm provides the logical steps and Python code applies the correct syntax to achieve the same result for N=6

2. Factorial Program

Written Algorithm:

```
BEGIN
  INPUT n
  SET factorial = 1
  FOR i = 1 TO n DO,
     factorial = factorial * i
  END FOR
  PRINT factorial
END
```

Python Implementation:

factorial.py - C:\Users\Dell\Desktop\HND\programming assignment hnd\factorial.py (3.13.5)

File Edit Format Run Options Window Help

```
# Factorial Program
n = 6
factorial = 1

print("\nFactorial Calculation for n =", n)

for i in range(1, n + 1):
    factorial = factorial * i
    print(f"After multiplying by {i}: factorial = {factorial}")

print("Final Factorial:", factorial)
```

Figure 11: Python implementation of the Factorial algorithm (N = 6)

```
Factorial Calculation for n = 6
After multiplying by 1: factorial = 1
After multiplying by 2: factorial = 2
After multiplying by 3: factorial = 6
After multiplying by 4: factorial = 24
After multiplying by 5: factorial = 120
After multiplying by 6: factorial = 720
Final Factorial: 720
```

Figure 12: Output of factorial program in Python Shell (Author developed, 2025)

Like the Fibonacci program, the pseudocode and Python implementation for the factorial calculation are also closely related. The pseudocode describes how to initialize a variable called factorial to 1, then using a loop to multiply it by each number from 1 to N, and finally displaying the result.

The Python code follows the exact same steps using factorial = 1 and a for loop for i in range(1, n+1): factorial = factorial * i. The initialization step, iteration, multiplication, and output match line by line between the pseudocode and Python code.

For instance, the pseudocode statement “factorial = factorial * i” matches precisely to the Python line factorial *= i, showing that the logic remains the same but is translated into proper Python syntax. This direct relationship shows how a well written algorithm acts as a blueprint for the Python program, ensuring correct logic and making the coding stage faster and error free.

Overall, the above screenshots and explanations clearly show how the pseudocode for the Factorial and Fibonacci algorithms corresponds to their Python implementations, each logic step in the pseudocode matches to a specific line or structure in Python code.

Programmers can reduce errors during coding phase by focusing on the logical flow first when using pseudocode as a planning tool, the same logic is then implemented in Python code by simply applying the right syntax which makes the development process faster, improves debugging and reliability, this clear relationship between algorithm and implementation is essential for structured program development.

Activity 02

2.1 Programming Paradigms

Explanation of a Programming Paradigm

A programming paradigm refers to the style of programming (GeeksforGeeks, 2023). It gives a set of rules, characteristic features and methods that guide how a programmer thinks about solving problems and how the program is structured. There is no single best way to solve every problem which is why program paradigms exist and different types of problems need different solutions.

For example, we can build a small calculator program with simple step by step instructions, but developing a large application like a social media platform needs a more complex structure such as objects, classes and events. With the help of paradigms, programmers can select the most efficient way to design and write code based on the project's needs as paradigms.

Main programming paradigms

A programming paradigm is a style of programming that provides a structure of how code is written and organized, different paradigms offer different approaches to solving problems and programmers choose the most suitable one based on the task. The below are some of the most commonly used paradigms.

- Procedural programming- gives the computer step by step instructions on how to do the task and in this the control flow is explicit meaning that the programmers can predict the flow of output, example, writing loops and conditions.
- Object Oriented Programming (OOP) – structures the code into classes and objects, which group data and behavior together (TutorialsPoint, 2023). For example, Java or Python using classes like Car or Student.
- Declarative programming- focuses on what the result should be and not how to achieve it, therefore the control flow is implicit as the flow of result cannot be predicted. For instance, SQL (Structured Query Language).

- Logic programming- in this rules and facts are used in order to find conclusions and solutions. Example for this is, Prolog language for AI problem solving.
- Event-Driven Programming- this style of programming is based on events, and the program waits for something to happen such as a mouse click, key press or screen touch. The program reacts to the event when it happens. For example, in a music app like Spotify, the song only plays when you click it.
- Functional Programming- Functional programming- builds programs using small math-like functions that do not change the data you give them, but returns a new result instead.

Characteristics of Procedural, Object-Oriented, and Event-Driven Paradigms

One of the oldest paradigms is the procedural programming and is based on giving the computer a step-by-step instruction to follow. In this, programs are broken down and divided into smaller parts called functions or procedures, which makes the code easier to read and reuse. This method uses loops, conditions and sequences to manage the control flow while maintaining a separation between data and functions. For example, in a simple Python program that prints the numbers from 1 to 5 using a loop, the instructions are written step by step and the program follows them in order to give a clear and predictable output, therefore the control flow is explicit meaning that the order of execution is direct and predictable.

On the other hand, Object-Oriented programming organizes the code according to objects. Objects are created from classes and combine both data known as attributes and actions known as methods. This paradigm supports reusability and flexibility in code by using important principles such as inheritance, polymorphism and encapsulation. Moreover, with OOP it is easier to model these into real world, for instance, we can create a car class in Python with data like brand and color, and methods such as startEngine(), this way the car is treated like a real-world object and the code becomes easier to understand, reuse and manage.

Whereas, Event-driven programming works differently from both procedural and object-oriented because the flow of the program is controlled by events rather than a fixed sequence of instructions. The program waits for something to happen such as a mouse click,

key press, or a sensor signal and then responds by executing the relevant event handler, this style is commonly used in graphical user interface (GUI) applications and interactive systems. For example, in a simple car racing game, the car only moves forward when the user press the up-arrow key, this demonstrates event-driven programming because the program waits for the user's action which is like the key press and then responds to it.

2.2 Advantages and Disadvantages of Programming Paradigms

Procedural Programming

Advantages:

Procedural programming is more suitable for beginners because it is simple and easy to understand. It uses step-by-step approach which ensures that the program's flow is predictable, clear and easy to follow. This linear execution model also makes debugging simple as each instruction is executed in sequence. Additionally, procedural programming works well for small programs or tasks where modularity is not a major concern.

Disadvantages:

Even though procedural programming is simple and straightforward, as programs get bigger and complicated it can be challenging to handle. The functions which are often designed for specific tasks restricts the reusability and maintainability of the program. Additionally, Large systems can be challenging to scale and modifications may need extensive changes throughout the codebase, making it less suitable for long term and complex projects

Object-Oriented Programming (OOP)

Advantages:

Through the use of classes and objects, object-oriented programming encourages modular and structured code that allows developers to efficiently separate functionality, and it enhances maintainability since changes made to one class generally have very less impact on others. Moreover, OOP improves data security through abstraction and encapsulation and

allows code reuse through inheritance and polymorphism, because of these characteristics, it is perfect for creating complex projects where maintainability and structure are essential.

Disadvantages:

OOP can be more challenging to learn when compared to procedural programming which makes it less suitable for beginners. Performance of the program may also be slower as a result of additional abstraction layer. Furthermore, OOP needs careful planning and design before implementation and it can be too complex for a simple project.

Event-Driven Programming

Advantages:

Event-driven programming is very effective for interactive applications, such as graphical user interfaces, web applications, and games, where the program responds to user inputs or system events. It works well with real-world situations as actions are triggered by events rather than sequential execution. This allows programs to easily handle multiple inputs simultaneously and respond in real time while improving user experience and the responsiveness of the system.

Disadvantages:

The non-linear flow of event driven programs can make tracing, debugging, and understanding program behavior more difficult. Additionally, event driven systems often use higher memory since multiple events may be monitored and handled in the background.

Also, developing large-scale event-driven applications requires careful planning of event handling mechanisms, which can be challenging for beginners or projects where simplicity and low resource usage are a must.

2. 3 Relationships among the Paradigms

Event-Driven vs Procedural Programming

Procedural programming follows a fixed sequence of instructions, where the flow is controlled by the programmer using functions and procedures. In contrast, event- driven programming focuses on responding to events such as a mouse click, key presses or system triggers, instead of executing code line by line, the program waits for events and runs the event handlers when those events occur. While procedural programming is more structured and the control flow is predictable, event-driven provides interactive and flexible control flow, which is commonly used in GUIs and real time applications.

Object-Oriented vs Procedural Programming

In order to solve problems, procedural programming uses function calls and step-by-step instructions to separate data and functions. Object-oriented programming (OOP) builds on these ideas by grouping data and behavior together inside objects and classes, this makes the program more organized and easier to manage as they grow in size. While procedural programming is simpler and works well for smaller tasks, OOP is more suitable for complex systems through features like encapsulation, inheritance, and polymorphism.

Event-Driven vs Object-Oriented Programming

Event driven programming focuses on handling events and triggering responses, whereas object-oriented programming helps to structure programs using objects and classes, these two paradigms also work together. For example, in a GUI application, an event like a click button is handled by a method inside a class, showing how event-driven behavior is organized using OOP structures. Event-driven defines when action happen while OOP defines how the program is organized to respond.

In conclusion, programming languages are not limited to a single paradigm in practice. Python for example, supports procedural, object-oriented and event-driven programming, allowing programmers to combine different approaches and techniques based on the problem they are trying to solve. Likewise, C# also supports multiple paradigms giving flexibility to developers, therefore, this overlap shows how different paradigms complement and support one another rather than being completely separate.

Comparison of Programming Paradigms:

(TutorialsPoint, 2023).

On the basis of	Procedural Programming	Object-Oriented Programming (OOP)	Event-driven Programming
Definition	Procedural programming gives the computer step by step instructions on how to do the task and in this the control flow is explicit meaning that the programmers can predict the flow of output	Object Oriented Programming (OOP) structures the code into classes and objects, which group data and behavior together	This style of programming is based on events and the program waits for something to happen such as a mouse click, key press or screen touch, then the program reacts to the event when it happens
Security	Data is global and shared so security is weaker.	Stronger security because of encapsulation	Security depends on how the events are handled, any improper handling may cause vulnerabilities.
Approach	Top-down approach, breaks the problem into functions.	Bottom-up approach, build objects first then combine.	Reactive approach, program waits and then responds to events.
Data movement	Data moves freely between functions but can cause errors if misused	Data is passed through objects	Data is triggered and handled when specific events occur
Reusability	The code can be reused but often requires rewriting functions thus reusability is limited	classes and objects can be reused in other programs so there is a high chance of reusability	event handlers can be reused but it depends heavily on the context, if the programmer wants to develop GUI systems or apps, so the reusability level is moderate
Real-world use	Can be used in simple programs and calculations	Used in complex systems like banking apps and enterprise software	Used in GUI apps, mobile games and interactive systems
Example	C and Pascal	Java, C++, Python	JavaScript, Visual Basic, C# with .NET, Node.js

Table 3: Comparison Table of different Programming Paradigms

In conclusion, each paradigm has its unique characteristics. Modern languages like Python will support multiple paradigms, allowing developers to mix procedural steps, object-oriented structures and event-driven behavior in one application, and this provides them with flexibility to select the best approach for each part of a project.

2.3 Critical Evaluation of Code Samples

Procedural Programming

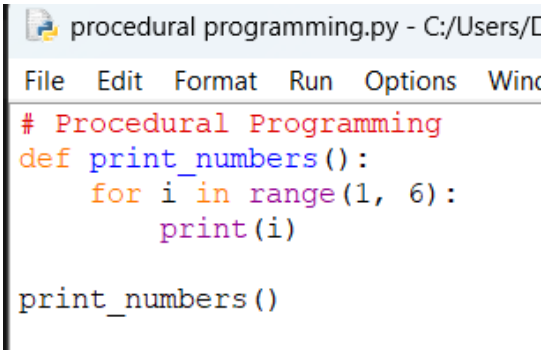


Figure 13: Procedural Programming Example (Author developed, 2025)

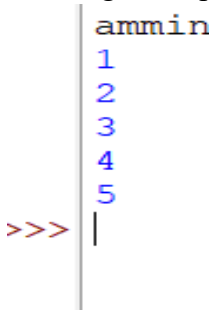


Figure 14: Procedural Programming Output (Author developed, 2025)

This code clearly shows the procedural programming paradigm, because the structure is straightforward and simple, a loop is included in the function print_numbers() and the program follows step-by-step instructions, from top to bottom. This reflects the control flow that procedural programming is known for, as the flow of execution is direct and can be easily predicted. Moreover, it is easy to read, write and execute, making it a good choice for small programs and beginner programming learners. The separation into a single function also shows the concept of procedures.

However, as the applications grow larger, this structure becomes less flexible, less adaptable and very difficult to maintain. Data and behavior cannot be grouped together and the code can become repetitive, if not carefully organized (Educative, 2021). For example, if we

wanted to print numbers with different rules like even numbers only, we would need to keep adding more functions or conditions, which would reduce the amount of code that could be reused. In summary, procedural programming works well and can be used for small programs and applications, but lacks modularity needed for complex systems as it grows.

Object-Oriented Programming

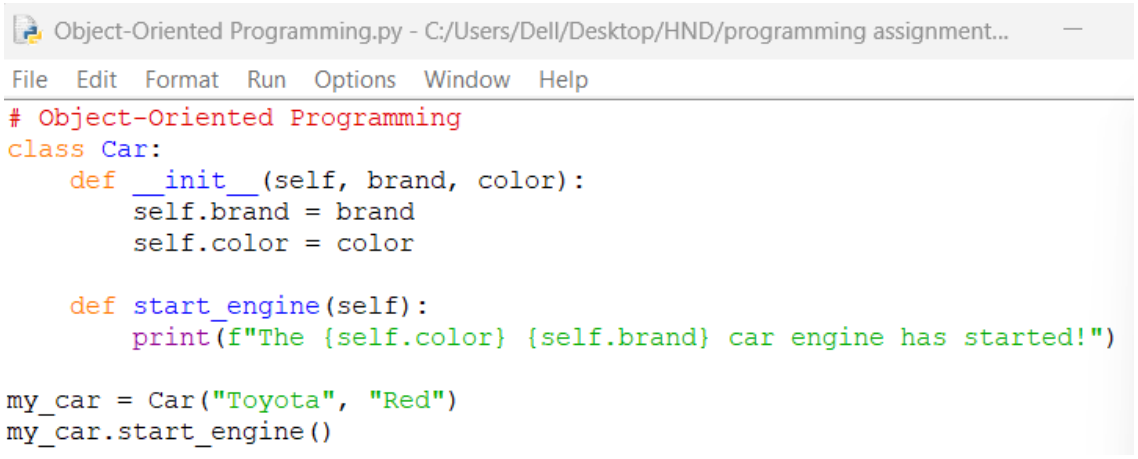


Figure 15: Object-Oriented Programming Example (Author developed, 2025)

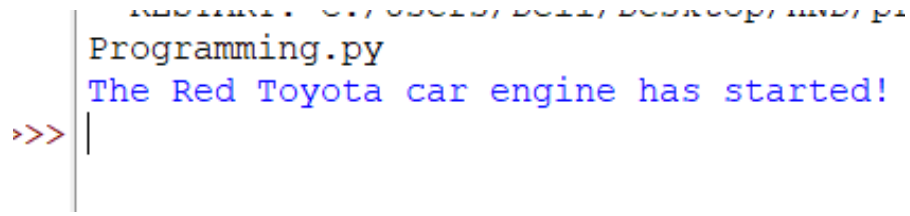


Figure 16: Object-Oriented Programming Output (Author developed, 2025)

This figure demonstrates the object-oriented paradigm, the structure is centered around a class (Car) which combines both data (attributes: brand, color) and behavior (method: start_engine). This shows OOP's unique characteristics of encapsulation (which keeps the related data and method together), and abstraction (hiding the internal details of how start_engine works).

The main advantage of this approach is that it makes the program more modular and reusable, for instance, we can easily create multiple Car objects with different attributes without having to rewrite the code again. It also makes the program easier to scale and maintain because responsibilities are clearly divided into classes.

On the other hand, OOP makes small programs more complicated and introduces extra complexity. Setting up classes and objects might feel unnecessary if the task is very simple. Additionally, badly designed OOP structures might result in complex hierarchies that are difficult to manage. In conclusion, OOP is powerful and flexible for medium to large systems, especially when the problem is used to solve real-world entities.

Event-Driven Programming

Event-Driven Programming.py - C:/Users/Dell/Desktop/HND/programming assignment hnc

File Edit Format Run Options Window Help

```
# Event-Driven Programming Example (using tkinter)
import tkinter as tk

def on_click():
    print("Button was clicked!")

window = tk.Tk()
button = tk.Button(window, text="Click Me", command=on_click)
button.pack()

window.mainloop()
```

Figure 17: Event-Driven Programming Example (Author developed, 2025)

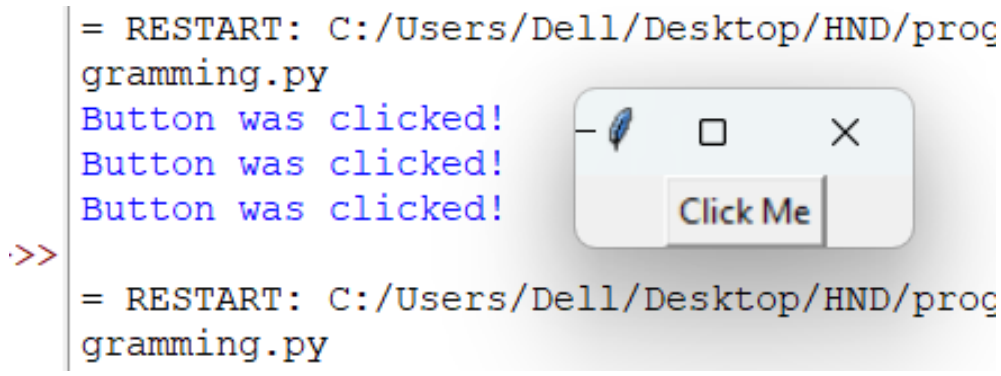


Figure 18: Event-Driven Programming Output (Author developed, 2025)

This is an example of an event-driven paradigm, in this the flow of the program is controlled by events. The structure is built around an event loop (window.mainloop()) that waits for user input, and a call function (on_click) that is triggered when the event (button press) occurs. This shows the unique characteristic of the event-driven programming, which

is the Inversion of control. Inversion of control is a special feature in which the program responds to user actions rather than determining the sequence in which it should execute. Additionally, this type of programming paradigm enables interactivity, making it ideal for GUIs, games, and apps where user input drives behavior, it also works naturally with modern application that must respond to unpredictable events.

On the contrary, event-driven programs can become harder to track down and debug, because the flow is no longer linear. If a developer does not keep track of which events triggers which handles, it might result in spaghetti code if not managed well. And it usually requires more setup compared to a simple procedural script. Therefore, event-driven programming works best for interactive applications and programs, but is less efficient for simply linear tasks, where procedural or OOP approaches are more suitable in that case.

To conclude, each programming paradigm has unique advantages and its own way of solving problems. Procedural programming is perfect for small tasks since it is easy to understand and use. Whereas, Object-oriented programming builds on this by adding structure and reusability, which makes it suitable for bigger programs. Event-driven programming on the other hand, is more focused on user interaction and is mainly used in apps and games where user actions control the flow. Therefore, no paradigm is best on its own, but modern languages like python and C# let developers to combine a mix of approaches, which provides them with flexibility and helps in building programs that are both efficient and easy to maintain.

In a real-world project like the KickBlast Judo system, these paradigms often work together, procedural logic handles basic calculations, OOP defines structured entities and event-driven programming manages GUI interactions, this combination shows that each paradigm has different strengths and the choice of paradigm depends on the specific problem and the design goals.

Activity 03

3.1 Pseudocode for calculating Monthly Training fees for KickBlast Judo

The following pseudocode shows the step-by-step process to calculate the monthly training fees for KickBlast, and it considers the training plan cost, competition fees (only for intermediate and Elite athletes), private coaching hours and then generates an itemized monthly total for each athlete, this ensures KickBlast Judo can manage athlete registrations and calculate the monthly payments accurately and effectively.

```
BEGIN
  // Step 1: Input athlete details
  INPUT athlete_name
  INPUT training_plan   // Beginner, Intermediate, Elite
  INPUT current_weight
  INPUT competition_weight_category
  INPUT num_competitions
  INPUT private_coaching_hours

  // Step 2: Set monthly training plan cost (weekly fee × 4)
  IF training_plan = "Beginner" THEN
     plan_cost ← 250 * 4
  ELSE IF training_plan = "Intermediate" THEN
     plan_cost ← 300 * 4
  ELSE IF training_plan = "Elite" THEN
     plan_cost ← 350 * 4
  ENDIF

  // Step 3: Validate competition entry
  IF training_plan = "Beginner" THEN
     num_competitions ← 0   // Beginners cannot compete
  ENDIF
  competition_cost ← num_competitions * 220
```

```
    // Step 4: Validate and calculate private coaching cost
    IF private_coaching_hours > 20 THEN
        private_coaching_hours ← 20  // max 5 hours/week × 4 weeks
    ENDIF
    private_coaching_cost ← private_coaching_hours * 90.50

    // Step 5: Calculate total monthly cost
    total_cost ← plan_cost + competition_cost + private_coaching_cost

    // Step 6: Output itemised costs
    DISPLAY "Athlete: ", athlete_name
    DISPLAY "Training Plan Cost (Rs): ", plan_cost
    DISPLAY "Competition Cost (Rs): ", competition_cost
    DISPLAY "Private Coaching Cost (Rs): ", private_coaching_cost
    DISPLAY "TOTAL Monthly Cost (Rs): ", total_cost

    // Step 7: Compare weights
    IF current_weight = competition_weight_category THEN
        DISPLAY "Athlete is in correct weight category."
    ELSE IF current_weight > competition_weight_category THEN
        DISPLAY "Athlete needs to reduce weight."
    ELSE
        DISPLAY "Athlete can increase weight to match category."
    ENDIF
END
```

Beginner Plan Dry Run Table:

Step	**Calculation / Action**	**Expected Outcome (Rs.)**	**Actual Outcome (Rs.)**
Training plan cost	250 × 4 weeks	1,000.00	1,000.00
Competition cost	Beginner → not	0.00	0.00

	allowed → 0 competitions		
Private coaching cost	5 hours × 90.50	452.50	452.50
Total monthly cost	1,000 + 0 + 452.50	1,452.50	1,452.50
Weight comparison	68 kg vs Lightweight category (73 kg)	can increase weight	can increase weight

Table 4: Dry Run Example Table for calculating Monthly Training fees (Author developed, 2025)

Final Output:

- Athlete: Kavindu
- Training Plan Cost: Rs. 1,000.00
- Competition Cost: Rs. 0.00
- Private Coaching Cost: Rs. 452.50
- Total Monthly Cost: Rs. 1,452.50
- Weight Note: Athlete can increase weight to match category.

The algorithm was also tested with Intermediate and Elite plans to ensure that competition costs and private coaching calculations are correctly applied across all training plans

3.2 C# Implementation of Monthly training fees program

```
// CALCULATE BUTTON

    private void btnCalculate_Click(object sender, EventArgs e)
    {
        try
        {
            if (!ValidateAllInputs())
                return;

            string trainingPlan = cmbTrainingPlan.Text;
            double currentWeight = double.Parse(txtCurrentWeight.Text);
            string weightCategory = cmbWeightCategory.Text;
            int numCompetitions = int.Parse(txtCompetitions.Text);
            double privateCoachingHours = double.Parse(txtPrivateCoaching.Text);

            planID = Convert.ToInt32(cmbTrainingPlan.SelectedValue);
            categoryID = Convert.ToInt32(cmbWeightCategory.SelectedValue);

            trainingPlanCost = CalculateTrainingPlanCost(planID);

            competitionCost = CalculateCompetitionCost(trainingPlan, ref numCompetitions);

            privateCoachingCost = CalculatePrivateCoachingCost(ref privateCoachingHours);

            totalMonthlyCost = CalculateTotalMonthlyCost(trainingPlanCost,
competitionCost,
                privateCoachingCost);

            double categoryLimit = GetWeightCategoryLimit(categoryID);
            weightStatus = CompareWeightWithCategory(currentWeight, categoryLimit);

            DisplayFeeBreakdown(txtAthleteName.Text, trainingPlan, currentWeight,
                weightCategory, numCompetitions, privateCoachingHours);

            btnSave.Enabled = true;

            MessageBox.Show("✓ Calculation completed successfully!",
                "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
        catch (Exception ex)
        {
            MessageBox.Show($"Calculation error: {ex.Message}",
```

```
            "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }
```

The above C# program allows KickBlast Judo to calculate monthly training fees, it also generates an itemized monthly cost and total fee after asking the user for the athlete's name, training schedule, contests and private coaching hours.

3.3 Screen shots of the developed database tables

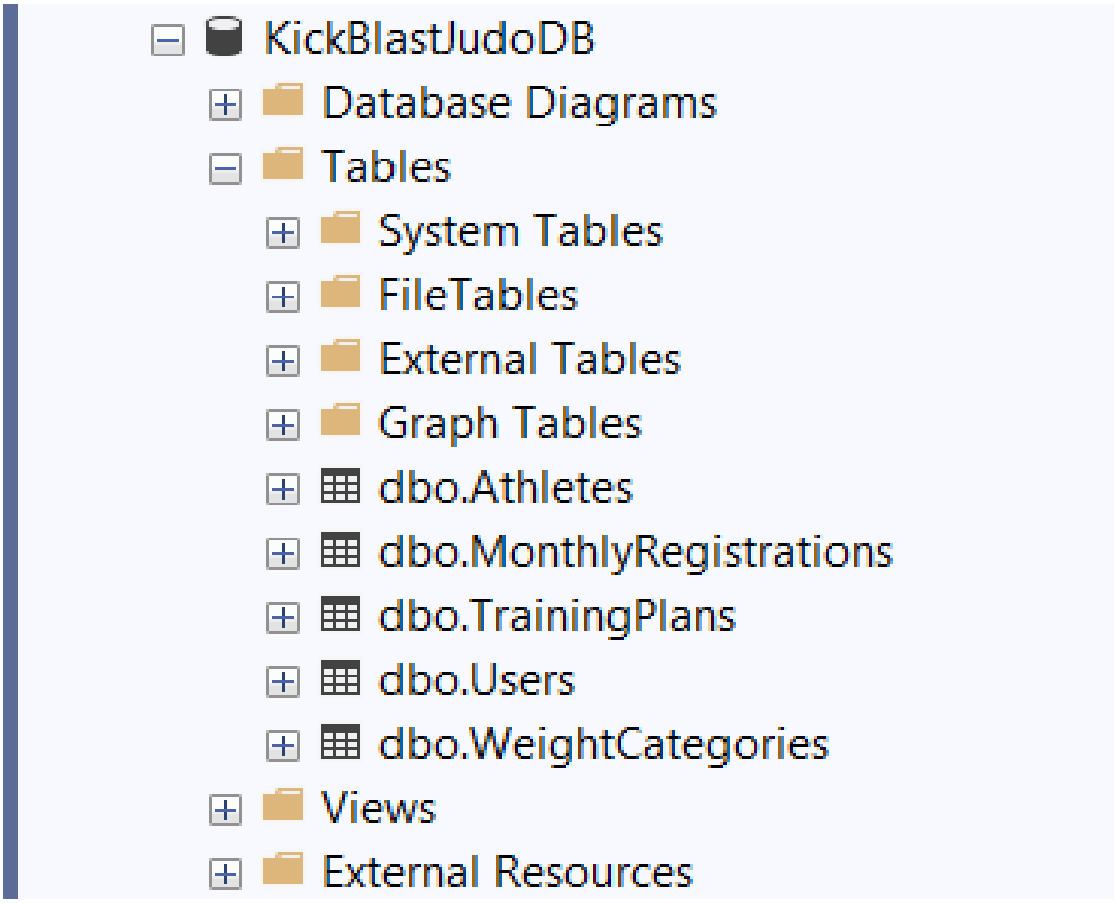


Figure 19: SQL database tables created for the KickBlast Judo System (Author developed, 2025)

3.4 Introduction to IDE (Integrated Development Environment)

Explanation of an IDE

An Integrated Development Environment (IDE) is a software application that brings together all the essential tools a programmer needs in one place (IBM, 2023). An IDE integrates the text editor, compiler, debugger and terminal into a single environment without the need for separate tools, this helps programmers to work more efficiently because they can write, test, and debug their code without constantly switching between different applications.

For example, when writing a program in C#, visual studio provides a code editor with color coding, suggestions for code completion, a built-in debugger and tools to run the program instantly, this makes development process faster, helps to reduce errors and allows programmers to focus more on solving problems rather than managing the tools.

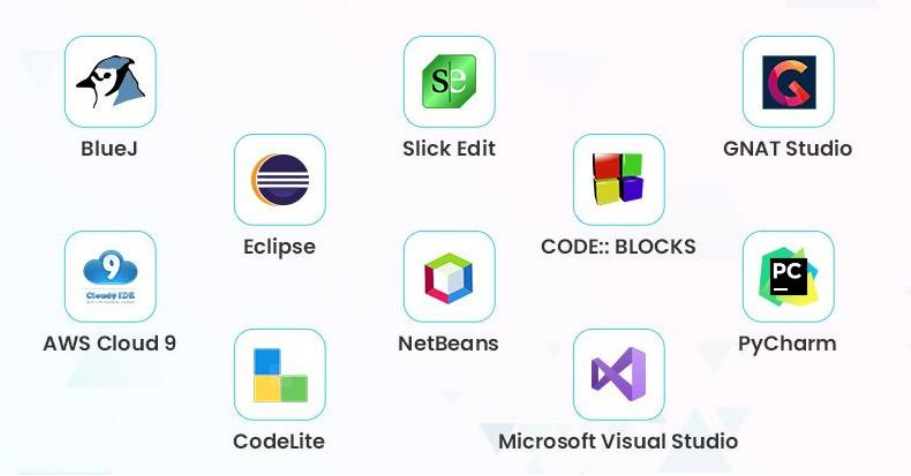


Figure 20: Examples of most commonly used IDEs (Jigar Shah, 2023)

For my KickBlast Judo System, I used Visual Studio IDE which helped me in designing the GUI, connecting the application with the database, writing the C# code and debugging the entire project in one environment.

Main components of an IDE and their importance

The three main components of an IDE are the source code editor, the program translator (compiler or interpreter), and the debugger.

1. Source Code Editor

The source code editor is the main area where programmers write their code, and it has useful features such as syntax highlighting, auto-completion and error indications that make coding easier and helps to prevent any common mistakes, a good source editor will improve the readability and allow developers to write code more efficiently and effectively. For example, when there are indentation errors or missing colons, the errors are highlighted immediately which helps fix issues before running the program and writing more larger codes.

When developing the KickBlast Judo system, this feature helped me quickly identify errors in C# syntax such as missing brackets or typos through color-coded text and IntelliSense suggestions.

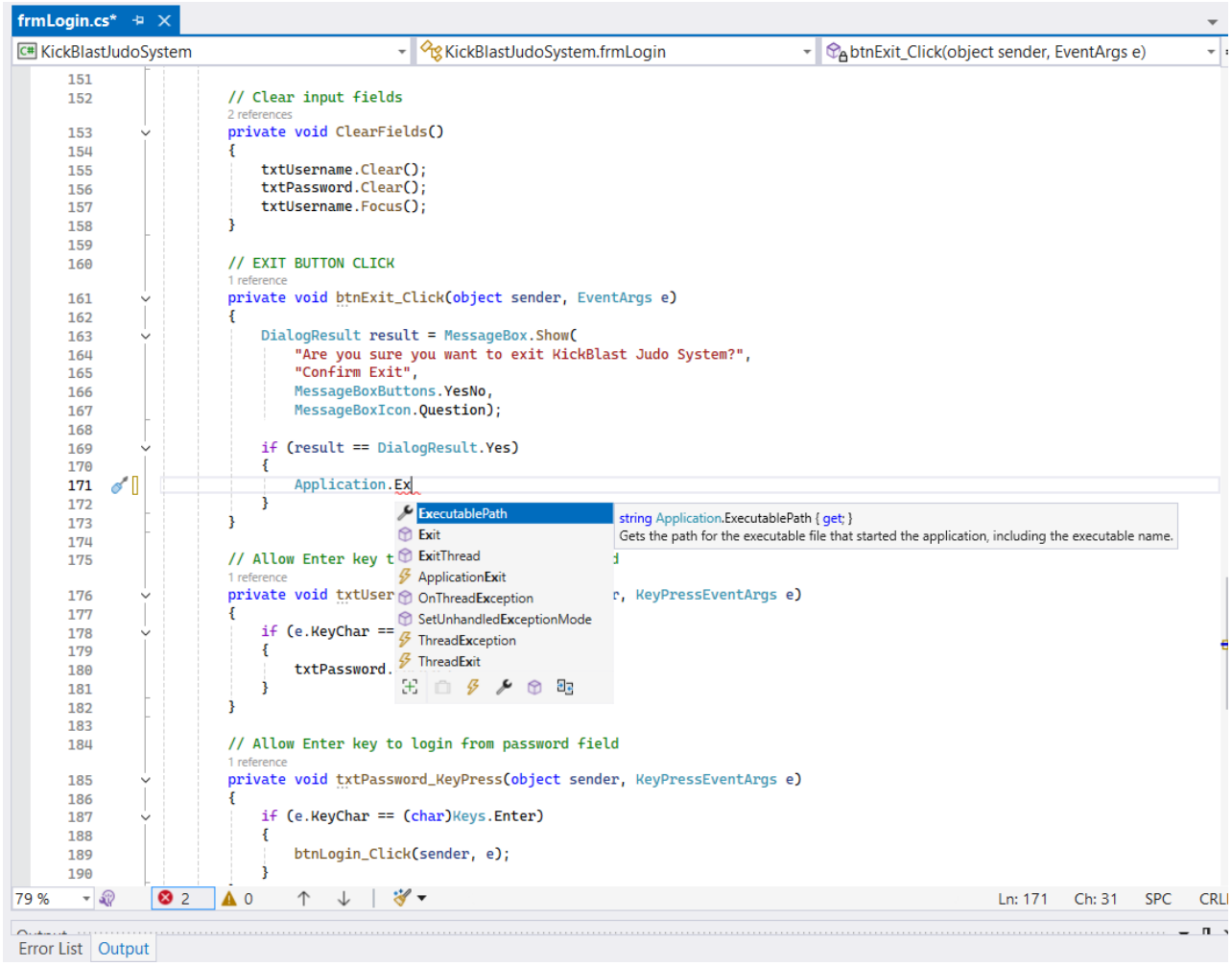


Figure 21: Visual Studio code editor showing IntelliSense suggestions while typing "Application.Exit()" (Author developed, 2025)

2. Program Translator

Once the code is written, it needs to be converted into a form where the computer can understand, this is done using a compiler or interpreter. A compiler translates the entire program into machine code before execution, while an interpreter translates and runs the code line by line, this component is essential because it allows the written source code to be executed as a working and a functional program. For instance, Python uses an interpreter to run code directly, while C# uses a compiler to create an executable file.

3. Debugger

The debugger helps to identify and fix errors that may arise while a program is running (GeeksforGeeks, 2023). It allows developers to run code step by step, set breakpoints and check variable values to find exactly where a problem occurs. This tool is very helpful for fixing errors and ensuring that the program runs as expected. For example, if a loop produces an unexpected result, setting a breakpoint inside the loop can help find the issue quickly and helps the developers resolve the problem.

When developing the KickBlast Judo system, I used the debugger to monitor variables such as total fee calculations and athlete IDs, which helped me ensure that the program produced correct and accurate results.

Visual Studio IDE and its features

One of the most widely used IDEs is Visual Studio, especially for C# and .NET applications, it is mostly used in industry because of its large toolkit that the programmers can use in order to build even large programs easily. During the development of the KickBlast Judo training fee system, several features of Visual Studio IDE were used in order to speed up the development process and reduce errors. Some of the key features are explained below:

1. IntelliSense

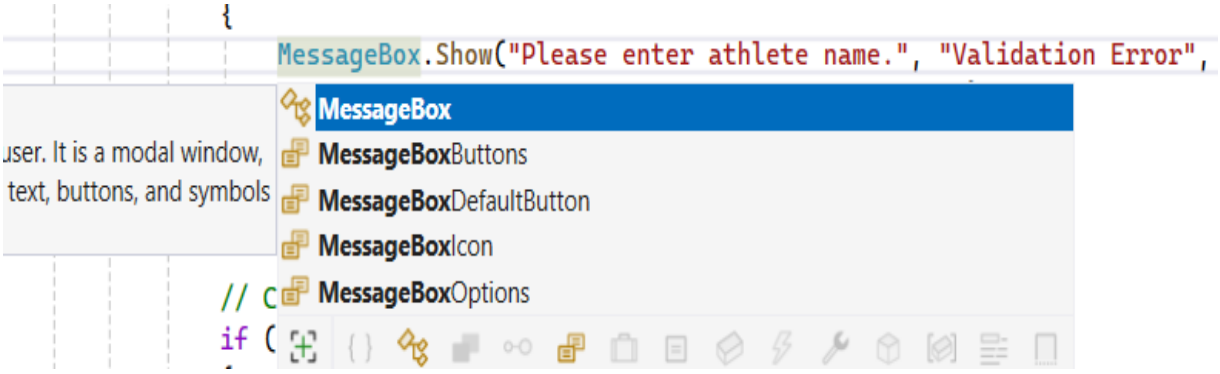


Figure 22: IntelliSense showing the correct syntax (Author developed, 2025)

IntelliSense is one of Visual Studio's most helpful features, which provides real-time code suggestions, auto-completion, parameter hints, and quick documentation as you type. For example, while typing methods like MessageBox.Show(), IntelliSense shows the correct syntax and available overloads, this feature helps to prevent any spelling errors and saves time because programmers do not have to memorize every method or property name and also helps beginners learn programming more quickly.

2. Debugging Tools

The Visual Studio debugger is extremely powerful, which helps developers find and fix errors by letting them pause the program at specific lines, step through the code, and view the current values stored in variables during execution. This makes it easier to detect calculation mistakes, find logic errors or unexpected behavior in the program.

For my program, I used the debugger to check the calculation of private coaching costs, I placed a breakpoint on the following line (line 119):

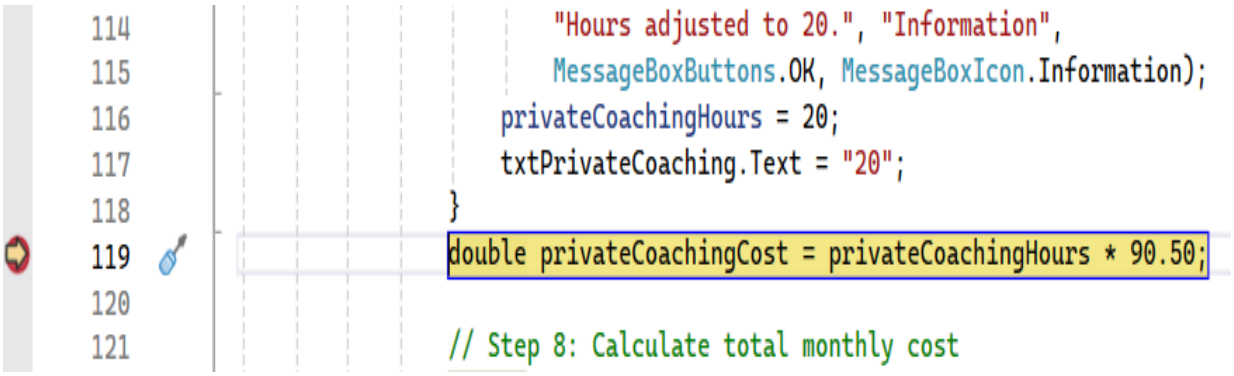


Figure 23: Debugger highlighted in yellow (Author developed, 2025)

```
116             privateCoachingHours = 20;
117             txtPrivateCoaching.Text = "20";
118         }
119         double privateCoachingCost = privateCoachingHours * 90.50;
120
121         // Step 8: Calculate total monthly cost
122         double totalCost = planCost + competitionCost + privateCoachingCost;
123
124         // Step 9: Compare weights
125         string weightComparison = CompareWeight(currentWeight, categoryLimit);
```

Figure 24: Breakpoints (Author developed, 2025)

When I ran the program in Debug Mode, the execution paused on this line and the debugger highlighted it in yellow, showing exactly where the program had stopped. At this point, I could see the value of **privateCoachingHours** in the Locals window to make sure it was being read correctly from the text box and confirm that the calculation would give the right result before the program continued. This process allowed me to check the accuracy of the calculations, find any issues early and make sure the program's logic worked as I expected.

3. Drag-and-Drop Form Designer

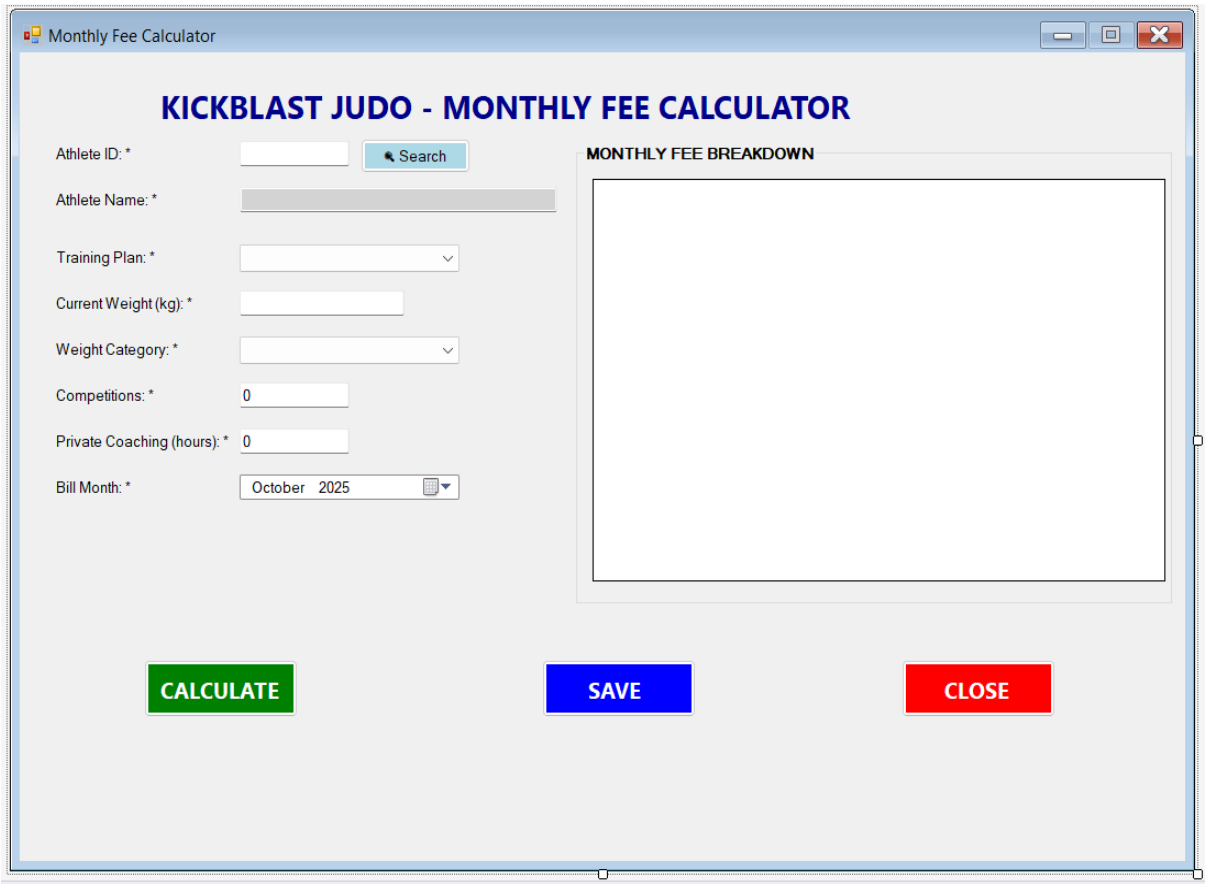


Figure 25: Windows Forms Designer (Author developed, 2025)

Moreover, the Windows Forms designer in Visual Studio allows developers to build GUI applications quickly using a drag-and-drop interface, instead of manually writing layout code, you can simply drag controls like Labels, TextBoxes, ComboBoxes, and Buttons onto the form, this feature made it easy to design the KickBlast interface especially placing input fields, output labels and buttons in a neat layout.

4. Solution Explorer

The Solution Explorer is a panel that displays all the files and resources in the Visual Studio project in a clear structure and it allows developers to easily switch between different forms, code files, resources and references. In this project, the Solution Explorer was used to quickly switch between the form design, like in (FrmLogin.cs [Design]). This made it easier to organize the entire KickBlast Judo system and keep track of multiple forms and codes within the same application.

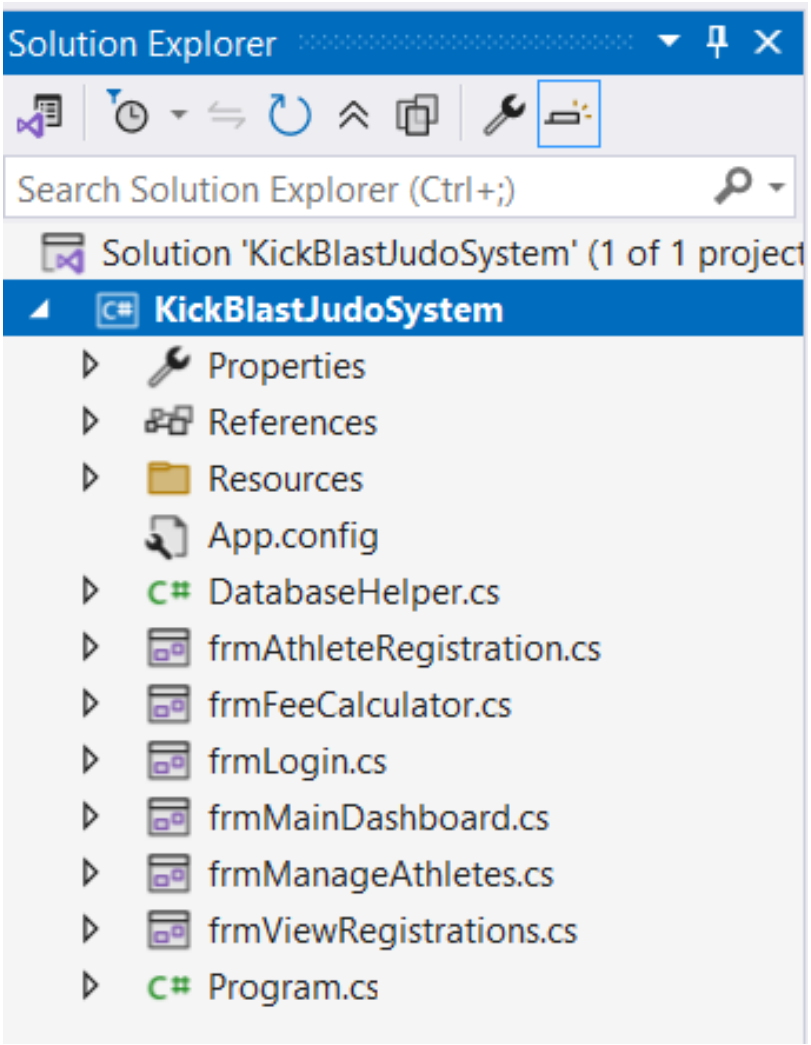


Figure 26: Solution Explorer showing the project structure and files (Author developed, 2025)

5. Output Window

The Output Window in Visual Studio is a powerful tool that shows real time messages during the execution of a system. In the KickBlast Judo project, it was very useful when testing the login form. When the "**Login**" button was clicked, the Output Window displayed a database connection error message because the SQL database had not been connected yet.

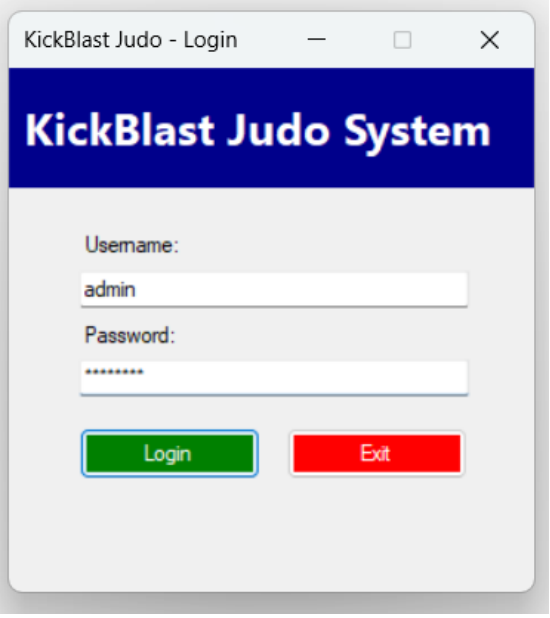


Figure 27: Output Window of Login Form (Author developed, 2025)

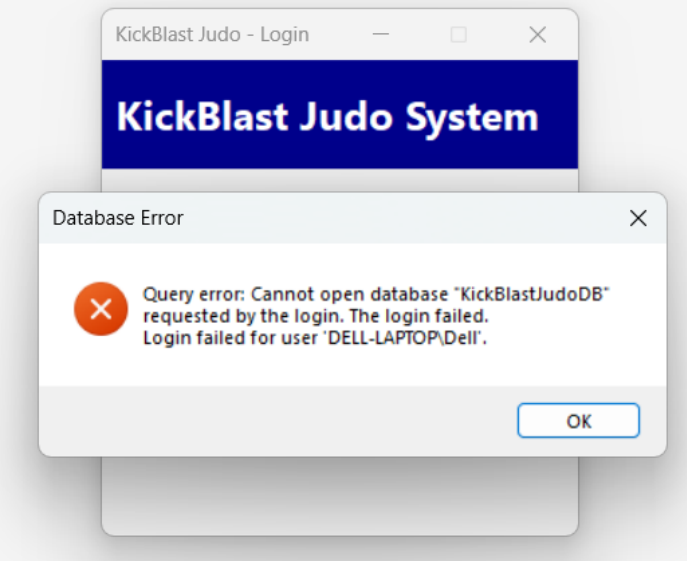


Figure 28: Output Window displaying a database error during login testing (Author developed, 2025)

This message clearly indicated where the issue was, allowing me to quickly identify that the problem was related to the database setup rather than the GUI or login logic. By reviewing

the error details shown in the Output Window, I was able to plan the next step for setting up and connecting the database correctly, this shows how the output window can be used for tracing errors and make debugging more efficient.

Evaluation of developing with and without an IDE

When developing a software application, programmers have to choose whether to use an Integrated Development Environment (IDE) or to code using more traditional tools like text editors and command line compilers, as both approaches can be used in order to create fully functional applications. In this evaluation, the advantages and disadvantages of developing without an IDE and with an IDE such as Visual Studio are compared against each other.

1. Developing with an IDE

Using an IDE such as Visual Studio provides a complete workspace that supports all type of software development stages, from writing the code, designing the interface to debugging and testing. For the KickBlast Judo system, which required multiple forms, input validation and fee calculations, Visual Studio offered several features that improved both productivity and quality.

One of the main benefits of using an IDE is IntelliSense (Microsoft, 2024). It is a built-in code completion tool that provides suggestions for syntax, variable names and methods all while typing, this feature helps to reduce human error and saves time by preventing typing mistakes. When developing the KickBlast Judo system, this was very useful to implement multiple event handlers and database connections as IntelliSense helped ensure the syntax and parameters were correct each time.

Another important feature is the debugger and visual studio allows the developer to set breakpoints, step through code line by line and check variable values at runtime, this type of interactive debugging method helps to identify logical or runtime errors quickly. For example, during the development of the Fee Calculator form in KickBlast Judo, the debugger made it easier to detect calculation mistakes in the training plan cost and display accurate results to users, without these features, finding the errors would have been extremely hard and might have taken significantly longer.

Additionally, Visual Studio integrates tools for project management, GUI design and version control all in one place, the Windows form Designer made it easy to visually arrange buttons, labels, and text boxes, allowing the GUI to be designed in a drag-and-drop manner rather than manually coding the interface elements. This feature was super helpful and valuable for creating a user-friendly interface for the application's dashboard and registration forms.

Overall, using Visual Studio improved the workflow by reducing manual tasks and offer real-time error detection and GUI creation, these benefits made the development of the KickBlast Judo application much faster, more organized and professional.

2. Developing without an IDE

On the other hand, when developing a software without an IDE such as by using basic text editor such as Notepad or Notepad++ and compiling code manually using the command line provides a completely different experience for the developers. This way of writing the code means that the developer must have a deep understanding of how code is written, compiled and executed.

One of the main benefits of this approach is that it helps developers gain a better understanding of programming fundamentals. When compiling from the command line, the developer must manually write and execute compilation commands, manage file paths and interpret compiler messages, this gives them a deeper understanding of how source code is converted into executable programs. As programmers learn to manage the entire development process without the use of automated tools, it also promotes independence.

Moreover, text editors are lightweight and need fewer system resources, this means that they can be used on any basic computer without large amounts of memory and high processing power. Also, this can be ideal for beginners who are learning basic syntax or for developers working in a much lower resource environment.

However, when developing larger and more complex projects, without features like debugging, IntelliSense, GUI design tools, the development process becomes slower and there can be many errors as well. It can be time consuming and ineffective to use command

line debuggers or manually inserting print statement to find logical and runtime errors. Additionally, managing multiple source files, images and resources can be very difficult without an organized project explorer.

If the KickBlast Judo system had been developed without an IDE, creating its graphical interface and testing its functions would have been very challenging. For example, when positioning controls like text boxes, buttons and labels would have needed manual coding of coordinates and layout management, this process would take much longer and leave room for visual inconsistencies. Similarly, without an integrated debugger, it would have been much harder to identify why a certain calculation or database connection failed.

When comparing both methods, the decision depends on the developer's goal, context of the project and which approach provides them the most benefits.

For educational purposes or small projects, developing without an IDE can be useful because it helps students understand what actually happens behind the code when it is compiled and executed, and it teaches valuable technical discipline and builds problem solving skills through manual debugging and file management. However, for more professional projects or applications which has user interfaces and database connections, using an IDE like Visual Studio is very efficient and effective. The built-in features allow developers to focus on logic and functionality rather than repetitive technical setup and the automated tools reduces the time spend on development process, improve code quality and help maintain consistency across multiple forms.

For the KickBlast Judo application, using Visual Studio made the project functional and successful. Moreover, it made it easier to design user-friendly forms for registration, display fee breakdowns, and validate inputs efficiently. The debugging tools ensured that all calculations and data handling worked correctly before the final execution. Since the system has event-driven interactions and database connectivity using Visual Studio's built-in features was very beneficial as well.

In conclusion, the evaluation shows that while developing without an IDE can be a valuable learning experience, using an IDE like Visual Studio offers a lot of practical advantages for modern application development. It increases productivity and reduces human error through

its intelligent tools and graphical design capabilities. For a complex project like the KickBlast Judo system, which involves multiple forms, data validation, and calculations, the use of Visual Studio IDE improved development efficiency and the overall quality of the final product. Therefore, it can be concluded that while manual coding environments provide strong foundational skills for learning and understanding, an IDE such as Visual Studio transforms the development process into a more efficient, accurate, and professional experience, mostly for applications designed for real-world use.

Activity 04

4. 1 Design and Implementation of the KickBlast Judo GUI system with Database Integration

Login Form GUI

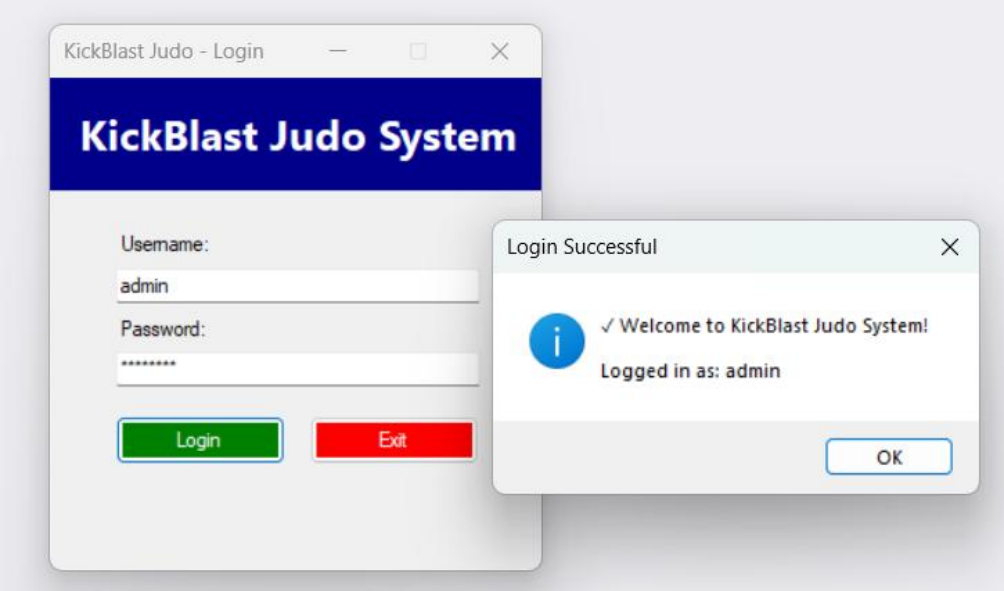


Figure 29: Login Interface (Author developed, 2025)

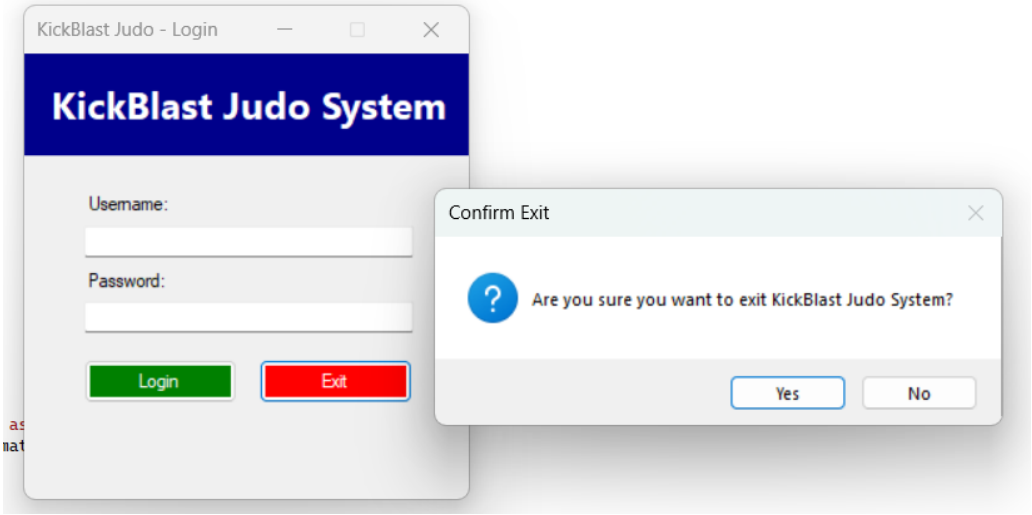


Figure 30: Exiting login interface (Author developed, 2025)

Login Form Code

```
using System;
using System.Data;
```

```
using System.Data.SqlClient;
using System.Windows.Forms;

namespace KickBlastJudoSystem
{
    public partial class frmLogin : Form
    {
        public static string LoggedInUser = "";

        public frmLogin()
        {
            InitializeComponent();
        }

        private void frmLogin_Load(object sender, EventArgs e)
        {
            // Test database connection
            if (!DatabaseHelper.TestConnection())
            {
                MessageBox.Show("⚠ Cannot connect to database!\n\n" +
                    "Please check:\n" +
                    "1. SQL Server is running\n" +
                    "2. Database 'KickBlastJudoDB' exists\n" +
                    "3. Connection string is correct",
                    "Database Connection Error",
                    MessageBoxButtons.OK, MessageBoxIcon.Error);
            }

            txtUsername.Focus();
        }

        // LOGIN BUTTON CLICK
        private void btnLogin_Click(object sender, EventArgs e)
        {
            if (!ValidateInputs())
                return;

            string username = txtUsername.Text.Trim();
            string password = txtPassword.Text;

            if (AuthenticateUser(username, password))
            {
                UpdateLastLogin(username);
```

```
            LoggedInUser = username;

            MessageBox.Show($"✓ Welcome to KickBlast Judo System!\n\nLogged in as:
{username}",
                "Login Successful", MessageBoxButtons.OK, MessageBoxIcon.Information);


            this.Hide();
            frmMainDashboard dashboard = new frmMainDashboard();
            dashboard.ShowDialog();

            this.Show();
            ClearFields();
            LoggedInUser = "";


            ClearFields();
            MessageBox.Show("Dashboard will open here after we create it!",
                "Info", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
        else
        {
            MessageBox.Show("✗ Invalid username or password!\n\nPlease try again.",
                "Login Failed", MessageBoxButtons.OK, MessageBoxIcon.Error);
            txtPassword.Clear();
            txtPassword.Focus();
        }
    }

    // Validate user inputs
    private bool ValidateInputs()
    {
        if (string.IsNullOrWhiteSpace(txtUsername.Text))
        {
            MessageBox.Show("Please enter your username.", "Validation Error",
                MessageBoxButtons.OK, MessageBoxIcon.Warning);
            txtUsername.Focus();
            return false;
        }

        if (string.IsNullOrWhiteSpace(txtPassword.Text))
        {
            MessageBox.Show("Please enter your password.", "Validation Error",
                MessageBoxButtons.OK, MessageBoxIcon.Warning);
```

```
            txtPassword.Focus();
            return false;
        }

        return true;
    }

    // Authenticate user against database
    private bool AuthenticateUser(string username, string password)
    {
        try
        {
            string query = @"SELECT COUNT(*), FullName, Role
                            FROM Users
                            WHERE Username = @Username
                            AND Password = @Password
                            AND IsActive = 1
                            GROUP BY FullName, Role";

            SqlParameter[] parameters = {
                new SqlParameter("@Username", username),
                new SqlParameter("@Password", password)
            };

            DataTable dt = DatabaseHelper.ExecuteQuery(query, parameters);

            if (dt.Rows.Count > 0)
            {
                return Convert.ToInt32(dt.Rows[0][0]) > 0;
            }

            return false;
        }
        catch (Exception ex)
        {
            MessageBox.Show($"Login error: {ex.Message}", "Error",
                MessageBoxButtons.OK, MessageBoxIcon.Error);
            return false;
        }
    }

    private void UpdateLastLogin(string username)
    {
        try
```

```
        {
            string query = @"UPDATE Users
                        SET LastLogin = GETDATE()
                        WHERE Username = @Username";

            SqlParameter[] parameters = {
                new SqlParameter("@Username", username)
            };

            DatabaseHelper.ExecuteNonQuery(query, parameters);
        }
        catch
        {

        }
    }

    private void ClearFields()
    {
        txtUsername.Clear();
        txtPassword.Clear();
        txtUsername.Focus();
    }

    // EXIT BUTTON CLICK
    private void btnExit_Click(object sender, EventArgs e)
    {
        DialogResult result = MessageBox.Show(
            "Are you sure you want to exit KickBlast Judo System?",
            "Confirm Exit",
            MessageBoxButtons.YesNo,
            MessageBoxIcon.Question);

        if (result == DialogResult.Yes)
        {
            Application.Exit();
        }
    }

    private void txtUsername_KeyPress(object sender, KeyPressEventArgs e)
    {
        if (e.KeyChar == (char)Keys.Enter)
        {
            txtPassword.Focus();
```

```
        }
    }

    private void txtPassword_KeyPress(object sender, KeyPressEventArgs e)
    {
        if (e.KeyChar == (char)Keys.Enter)
        {
            btnLogin_Click(sender, e);
        }
    }
  }
}
```

Main Dashboard Form GUI

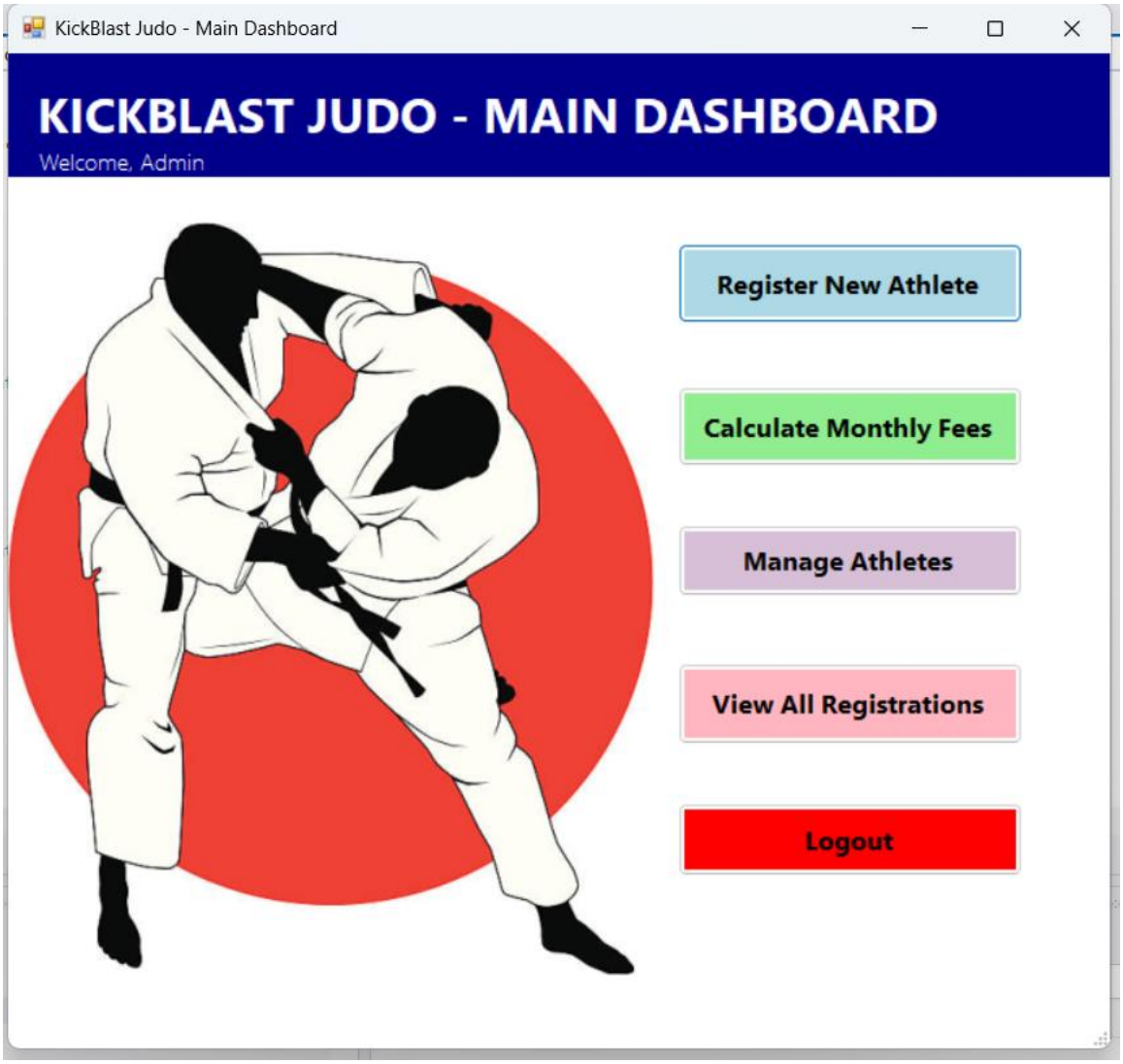


Figure 31: Main Dashboard of KickBlast Judo system (Author developed, 2025)

Main Dashboard Form Code

```
using System;
using System.Windows.Forms;

namespace KickBlastJudoSystem
{
   public partial class frmMainDashboard : Form
   {
      public frmMainDashboard()
      {
         InitializeComponent();
      }

      private void frmMainDashboard_Load(object sender, EventArgs e)
      {
         lblWelcome.Text = $"Welcome, {frmLogin.LoggedInUser}";
      }

      // Button 1: Register New Athlete
      private void btnAthleteReg_Click(object sender, EventArgs e)
      {
         frmAthleteRegistration regForm = new frmAthleteRegistration();
         regForm.ShowDialog();
      }

      // Button 2: Calculate Monthly Fees
      private void btnFeeCalculator_Click(object sender, EventArgs e)
      {
         frmFeeCalculator calcForm = new frmFeeCalculator();
         calcForm.ShowDialog();
      }

      // Button 3: View All Registrations
      private void btnViewRegistrations_Click(object sender, EventArgs e)
      {
         frmViewRegistrations viewForm = new frmViewRegistrations();
         viewForm.ShowDialog();
      }

      // Button 4: Manage Athletes
      private void btnManageAthletes_Click(object sender, EventArgs e)
      {
         frmManageAthletes manageForm = new frmManageAthletes();
```

```
            manageForm.ShowDialog();
        }

        // Button 5: Logout
        private void btnLogout_Click(object sender, EventArgs e)
        {
            DialogResult result = MessageBox.Show(
                "Are you sure you want to logout?",
                "Confirm Logout",
                MessageBoxButtons.YesNo,
                MessageBoxIcon.Question);

            if (result == DialogResult.Yes)
            {
                this.Close();
            }
        }
    }
}
```

Athlete Registration Form GUI

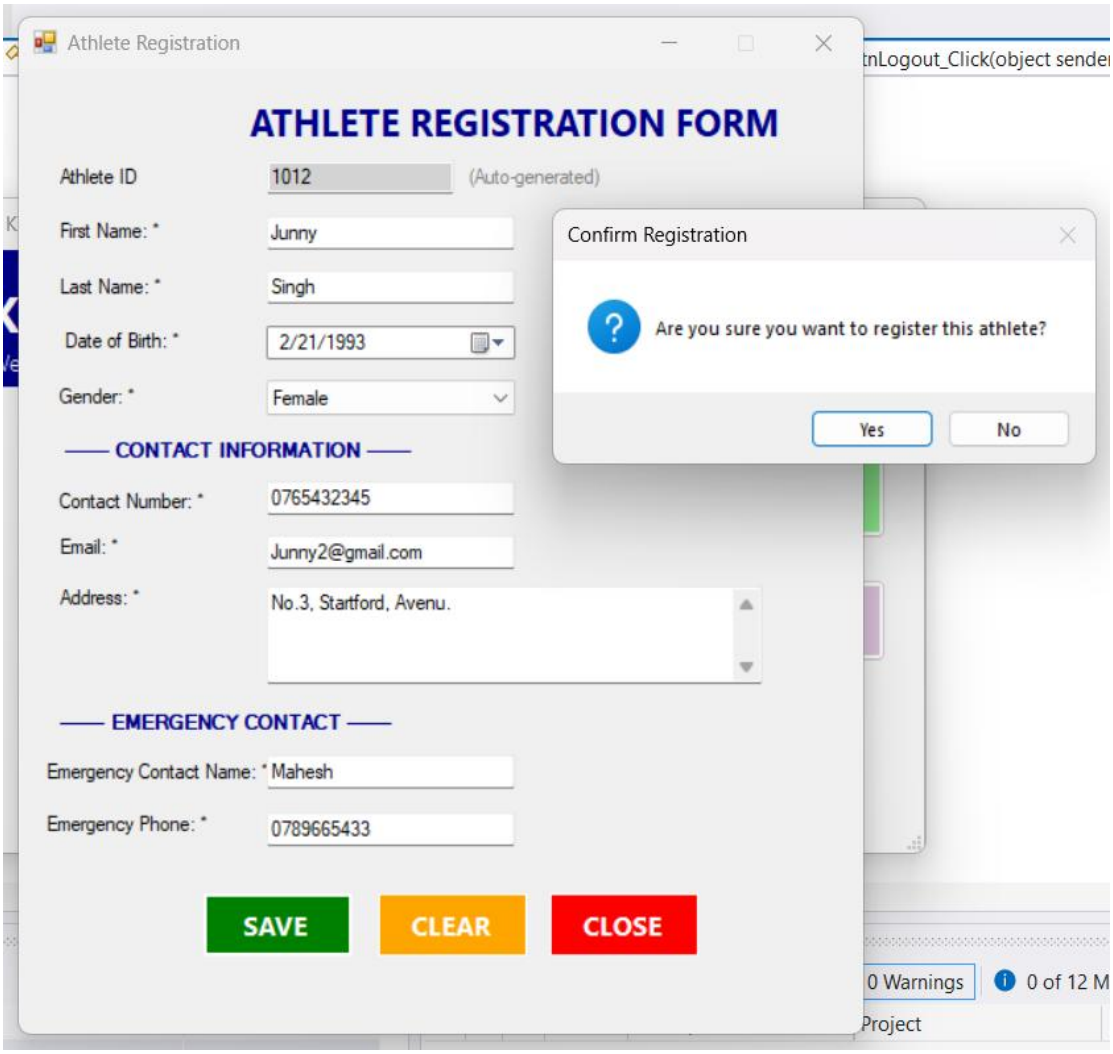


Figure 32: Athlete Registration form (Author developed, 2025)

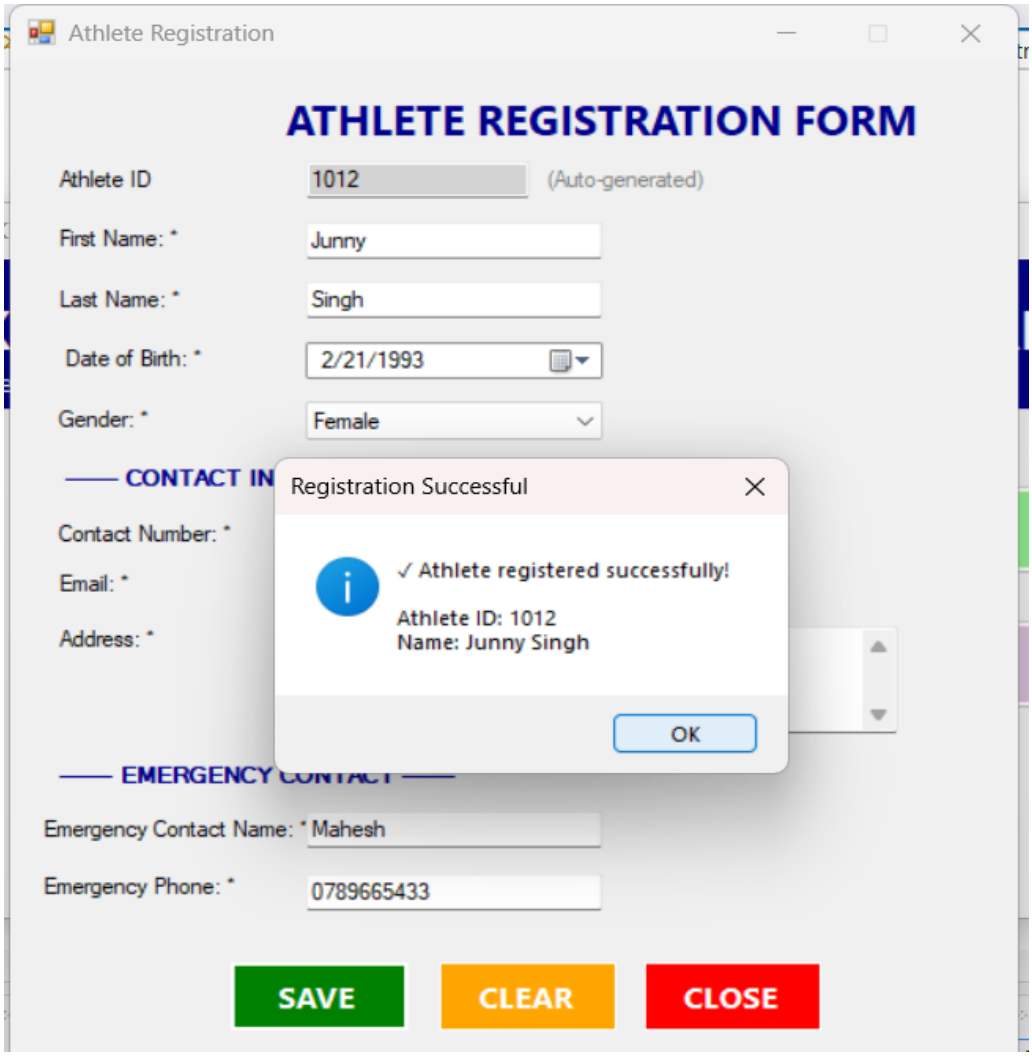


Figure 33: Athlete registered successfully in the system (Author developed, 2025)

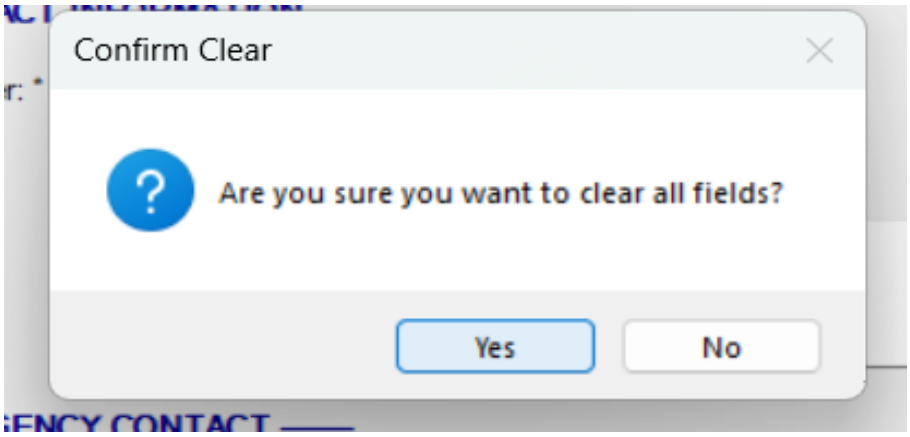


Figure 34: Clearing new registration details (Author developed, 2025)

Athlete Registration Form Code

```
using System;
using System.Data;
```

```
using System.Data.SqlClient;
using System.Windows.Forms;

namespace KickBlastJudoSystem
{
   public partial class frmAthleteRegistration : Form
   {
      public frmAthleteRegistration()
      {
         InitializeComponent();
         this.Load += frmAthleteRegistration_Load;
      }

      private void frmAthleteRegistration_Load(object sender, EventArgs e)
      {
         InitializeForm();
         LoadNextAthleteID();
      }

      private void InitializeForm()
      {

         dtpDOB.MaxDate = DateTime.Now.AddYears(-5);  // minimum 5 years old
         dtpDOB.MinDate = DateTime.Now.AddYears(-80); // maximum 80 years old
         dtpDOB.Value = DateTime.Now.AddYears(-15);   // default 15 years old


         if (cmbGender.Items.Count == 0)
         {
            cmbGender.Items.AddRange(new string[] { "Male", "Female", "Other" });
         }

         txtFirstName.Focus();
      }

      // Load next available Athlete IDs
      private void LoadNextAthleteID()
      {
         try
         {
            string query = "SELECT ISNULL(MAX(AthleteID), 1000) + 1 FROM Athletes";
            object result = DatabaseHelper.ExecuteScalar(query);

            if (result != null)
```

```
        {
            txtAthleteID.Text = result.ToString();
        }
        else
        {
            txtAthleteID.Text = "1001"; // Starting ID
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Error loading Athlete ID: {ex.Message}",
            "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
        txtAthleteID.Text = "1001";
    }
}

// SAVE BUTTON
private void btnSave_Click(object sender, EventArgs e)
{
    if (!ValidateInputs())
        return;

    DialogResult result = MessageBox.Show(
        "Are you sure you want to register this athlete?",
        "Confirm Registration",
        MessageBoxButtons.YesNo,
        MessageBoxIcon.Question);

    if (result != DialogResult.Yes)
        return;

    if (SaveAthleteToDatabase())
    {
        MessageBox.Show(
            $"✓ Athlete registered successfully!\n\n" +
            $"Athlete ID: {txtAthleteID.Text}\n" +
            $"Name: {txtFirstName.Text} {txtLastName.Text}",
            "Registration Successful",
            MessageBoxButtons.OK,
            MessageBoxIcon.Information);

        ClearAllFields();
        LoadNextAthleteID();
    }
```

```
    }

    private bool ValidateInputs()
    {
        if (string.IsNullOrWhiteSpace(txtFirstName.Text))
        {
            ShowValidationError("Please enter First Name.", txtFirstName);
            return false;
        }

        if (string.IsNullOrWhiteSpace(txtLastName.Text))
        {
            ShowValidationError("Please enter Last Name.", txtLastName);
            return false;
        }

        if (cmbGender.SelectedIndex == -1)
        {
            ShowValidationError("Please select Gender.", cmbGender);
            return false;
        }

        if (string.IsNullOrWhiteSpace(txtContact.Text))
        {
            ShowValidationError("Please enter Contact Number.", txtContact);
            return false;
        }

        if (txtContact.Text.Length < 10)
        {
            ShowValidationError("Contact Number must be at least 10 digits.", txtContact);
            return false;
        }

        if (string.IsNullOrWhiteSpace(txtAddress.Text))
        {
            ShowValidationError("Please enter Address.", txtAddress);
            return false;
        }

        if (string.IsNullOrWhiteSpace(txtEmergencyContact.Text))
        {
            ShowValidationError("Please enter Emergency Contact Name.",
txtEmergencyContact);
```

```
            return false;
        }

        if (string.IsNullOrWhiteSpace(txtEmergencyPhone.Text))
        {
            ShowValidationError("Please enter Emergency Contact Phone.",
txtEmergencyPhone);
            return false;
        }

        if (txtEmergencyPhone.Text.Length < 10)
        {
            ShowValidationError("Emergency Phone must be at least 10 digits.",
txtEmergencyPhone);
            return false;
        }

        return true;
    }

    // error message
    private void ShowValidationError(string message, Control control)
    {
        MessageBox.Show(message, "Validation Error",
            MessageBoxButtons.OK, MessageBoxIcon.Warning);
        control.Focus();
    }

    // Save athlete to database
    private bool SaveAthleteToDatabase()
    {
        try
        {
            string query = @"INSERT INTO Athletes
                (FirstName, LastName, DateOfBirth, Gender, ContactNumber, Email,
                 Address, EmergencyContactName, EmergencyContactPhone, CreatedBy)
            VALUES
                (@FirstName, @LastName, @DOB, @Gender, @Contact, @Email,
                 @Address, @EmergencyContact, @EmergencyPhone, @CreatedBy)";

            SqlParameter[] parameters = {
                new SqlParameter("@FirstName", txtFirstName.Text.Trim()),
                new SqlParameter("@LastName", txtLastName.Text.Trim()),
                new SqlParameter("@DOB", dtpDOB.Value.Date),
```

```
            new SqlParameter("@Gender", cmbGender.SelectedItem.ToString()),
            new SqlParameter("@Contact", txtContact.Text.Trim()),
            new SqlParameter("@Email", string.IsNullOrWhiteSpace(txtEmail.Text) ?
               (object)DBNull.Value : txtEmail.Text.Trim()),
            new SqlParameter("@Address", txtAddress.Text.Trim()),
            new SqlParameter("@EmergencyContact", txtEmergencyContact.Text.Trim()),
            new SqlParameter("@EmergencyPhone", txtEmergencyPhone.Text.Trim()),
            new SqlParameter("@CreatedBy", frmLogin.LoggedInUser)
        };

        int rowsAffected = DatabaseHelper.ExecuteNonQuery(query, parameters);

        return rowsAffected > 0;
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Error saving athlete:\n\n{ex.Message}",
            "Database Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
        return false;
    }
}

// CLEAR BUTTON
private void btnClear_Click(object sender, EventArgs e)
{
    DialogResult result = MessageBox.Show(
        "Are you sure you want to clear all fields?",
        "Confirm Clear",
        MessageBoxButtons.YesNo,
        MessageBoxIcon.Question);

    if (result == DialogResult.Yes)
    {
        ClearAllFields();
    }
}

private void ClearAllFields()
{
    txtFirstName.Clear();
    txtLastName.Clear();
    dtpDOB.Value = DateTime.Now.AddYears(-15);
    cmbGender.SelectedIndex = -1;
    txtContact.Clear();
```

```
            txtEmail.Clear();
            txtAddress.Clear();
            txtEmergencyContact.Clear();
            txtEmergencyPhone.Clear();
            txtFirstName.Focus();
        }

        // CLOSE BUTTON
        private void btnClose_Click(object sender, EventArgs e)
        {
            this.Close();
        }

        private void txtContact_KeyPress(object sender, KeyPressEventArgs e)
        {
            if (!char.IsDigit(e.KeyChar) && e.KeyChar != (char)Keys.Back)
            {
                e.Handled = true;
            }
        }

        private void txtEmergencyPhone_KeyPress(object sender, KeyPressEventArgs e)
        {
            if (!char.IsDigit(e.KeyChar) && e.KeyChar != (char)Keys.Back)
            {
                e.Handled = true;
            }
        }
    }
}
```

Monthly Fee Calculator Form GUI

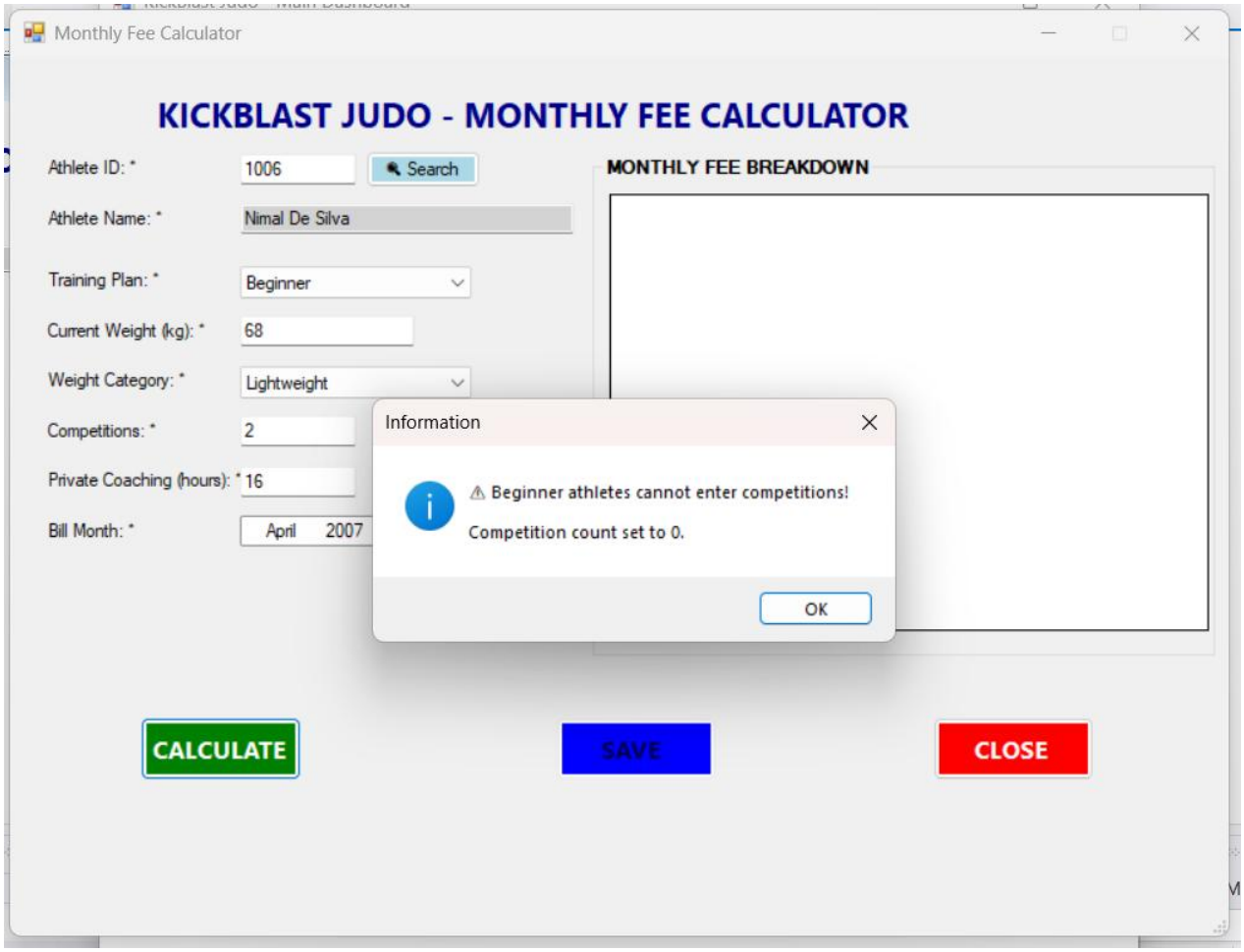


Figure 35: Monthly fee calculator (Author developed, 2025)

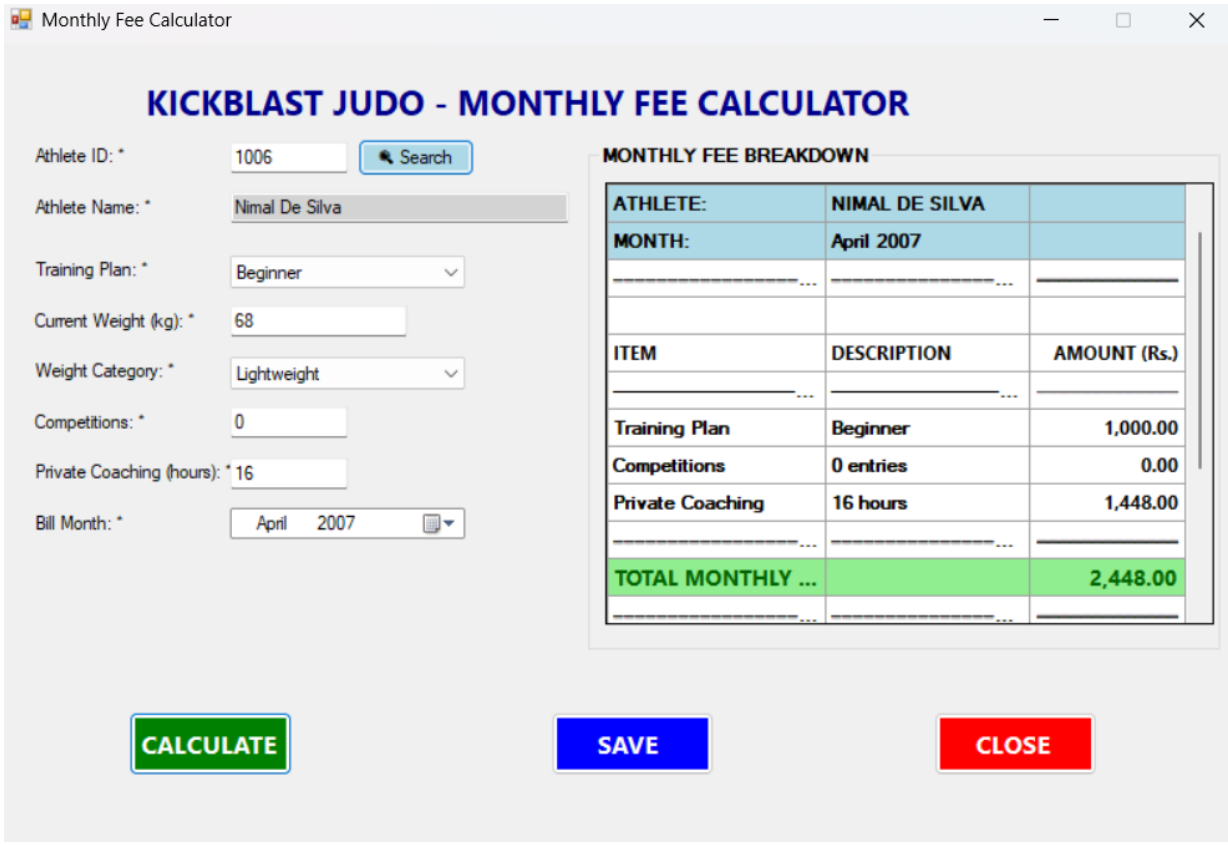


Figure 36: Calculated Total Monthly Cost (Author developed, 2025)

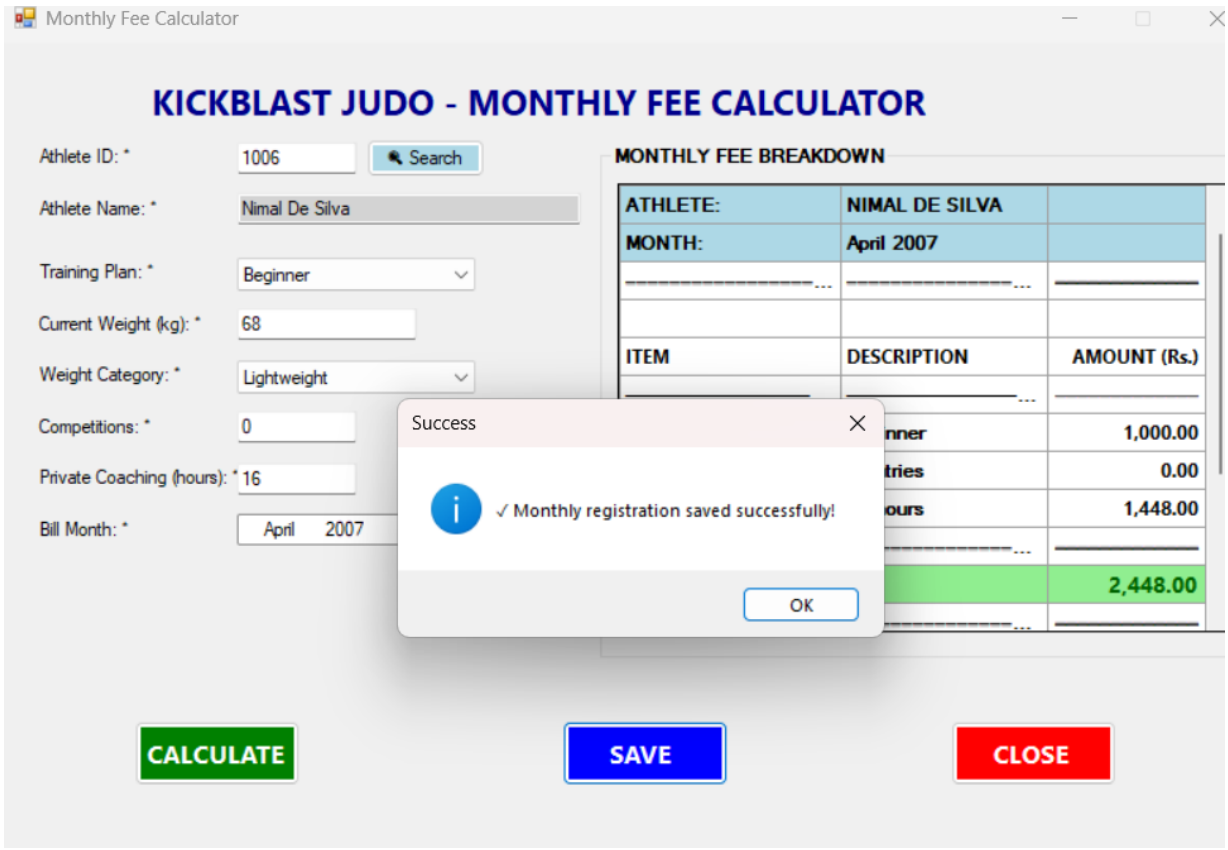


Figure 37: Monthly registration cost saved successfully (Author developed, 2025)

Monthly Fee Calculator Form Code

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Windows.Forms;

namespace KickBlastJudoSystem
{
   public partial class frmFeeCalculator : Form
   {
      private int athleteID = 0;
      private int planID = 0;
      private int categoryID = 0;
      private double trainingPlanCost = 0;
      private double competitionCost = 0;
      private double privateCoachingCost = 0;
      private double totalMonthlyCost = 0;
      private string weightStatus = "";

      public frmFeeCalculator()
      {
         InitializeComponent();
         this.Load += FrmFeeCalculator_Load;
      }

      private void FrmFeeCalculator_Load(object sender, EventArgs e)
      {
         InitializeFormDefaults();
         LoadTrainingPlans();
         LoadWeightCategories();
      }

      // Method 1: Initialize form defaults
      private void InitializeFormDefaults()
      {
         dtpMonth.Value = new DateTime(DateTime.Now.Year, DateTime.Now.Month, 1);
         txtAthleteID.Focus();
      }

      // Method 2: Load training plans from database
      private void LoadTrainingPlans()
      {
         try
```

```
        {
            string query = "SELECT PlanID, PlanName FROM TrainingPlans WHERE IsActive = 1
ORDER BY PlanID";
            DataTable dt = DatabaseHelper.ExecuteQuery(query);

            cmbTrainingPlan.DataSource = dt;
            cmbTrainingPlan.DisplayMember = "PlanName";
            cmbTrainingPlan.ValueMember = "PlanID";
            cmbTrainingPlan.SelectedIndex = -1;
        }
        catch (Exception ex)
        {
            MessageBox.Show($"Error loading training plans: {ex.Message}",
                "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }

    // Method 3: Load weight categories from database
    private void LoadWeightCategories()
    {
        try
        {
            string query = "SELECT CategoryID, CategoryName FROM WeightCategories WHERE
IsActive = 1 ORDER BY CategoryID";
            DataTable dt = DatabaseHelper.ExecuteQuery(query);

            cmbWeightCategory.DataSource = dt;
            cmbWeightCategory.DisplayMember = "CategoryName";
            cmbWeightCategory.ValueMember = "CategoryID";
            cmbWeightCategory.SelectedIndex = -1;
        }
        catch (Exception ex)
        {
            MessageBox.Show($"Error loading weight categories: {ex.Message}",
                "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }

    // SEARCH BUTTON
    private void btnSearchAthlete_Click(object sender, EventArgs e)
    {
        if (string.IsNullOrWhiteSpace(txtAthleteID.Text))
        {
            MessageBox.Show("Please enter Athlete ID.", "Validation Error",
```

```
            MessageBoxButtons.OK, MessageBoxIcon.Warning);
        txtAthleteID.Focus();
        return;
    }

    if (!int.TryParse(txtAthleteID.Text, out int id))
    {
        MessageBox.Show("Athlete ID must be a number.", "Validation Error",
            MessageBoxButtons.OK, MessageBoxIcon.Warning);
        txtAthleteID.Focus();
        return;
    }

    SearchAthleteByID(id);
}

// Method 4: Search and load athlete details
private void SearchAthleteByID(int id)
{
    try
    {
        string query = @"SELECT AthleteID, CONCAT(FirstName, ' ', LastName) AS FullName
                    FROM Athletes
                    WHERE AthleteID = @AthleteID AND IsActive = 1";

        SqlParameter[] parameters = {
            new SqlParameter("@AthleteID", id)
        };

        DataTable dt = DatabaseHelper.ExecuteQuery(query, parameters);

        if (dt.Rows.Count > 0)
        {
            athleteID = Convert.ToInt32(dt.Rows[0]["AthleteID"]);
            txtAthleteName.Text = dt.Rows[0]["FullName"].ToString();

            MessageBox.Show("✓ Athlete found!", "Success",
                MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
        else
        {
            athleteID = 0;
            txtAthleteName.Clear();
            MessageBox.Show("✗ Athlete not found!\n\nPlease check the Athlete ID.",
```

```
                "Not Found", MessageBoxButtons.OK, MessageBoxIcon.Warning);
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show($"Error searching athlete: {ex.Message}",
                "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }

    // CALCULATE BUTTON

    private void btnCalculate_Click(object sender, EventArgs e)
    {
        try
        {
            if (!ValidateAllInputs())
                return;

            string trainingPlan = cmbTrainingPlan.Text;
            double currentWeight = double.Parse(txtCurrentWeight.Text);
            string weightCategory = cmbWeightCategory.Text;
            int numCompetitions = int.Parse(txtCompetitions.Text);
            double privateCoachingHours = double.Parse(txtPrivateCoaching.Text);

            planID = Convert.ToInt32(cmbTrainingPlan.SelectedValue);
            categoryID = Convert.ToInt32(cmbWeightCategory.SelectedValue);

            trainingPlanCost = CalculateTrainingPlanCost(planID);

            competitionCost = CalculateCompetitionCost(trainingPlan, ref numCompetitions);

            privateCoachingCost = CalculatePrivateCoachingCost(ref privateCoachingHours);

            totalMonthlyCost = CalculateTotalMonthlyCost(trainingPlanCost, competitionCost,
                privateCoachingCost);

            double categoryLimit = GetWeightCategoryLimit(categoryID);
            weightStatus = CompareWeightWithCategory(currentWeight, categoryLimit);

            DisplayFeeBreakdown(txtAthleteName.Text, trainingPlan, currentWeight,
                weightCategory, numCompetitions, privateCoachingHours);

            btnSave.Enabled = true;
```

```
            MessageBox.Show("✓ Calculation completed successfully!",
                "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
        catch (Exception ex)
        {
            MessageBox.Show($"Calculation error: {ex.Message}",
                "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }


    // Method 5: Validate all inputs
    private bool ValidateAllInputs()
    {
        if (athleteID == 0)
        {
            ShowValidationError("Please search for an athlete first.", txtAthleteID);
            return false;
        }

        if (cmbTrainingPlan.SelectedIndex == -1)
        {
            ShowValidationError("Please select a Training Plan.", cmbTrainingPlan);
            return false;
        }

        if (!double.TryParse(txtCurrentWeight.Text, out double weight) || weight <= 0)
        {
            ShowValidationError("Please enter valid Current Weight.", txtCurrentWeight);
            return false;
        }

        if (cmbWeightCategory.SelectedIndex == -1)
        {
            ShowValidationError("Please select Weight Category.", cmbWeightCategory);
            return false;
        }

        if (!int.TryParse(txtCompetitions.Text, out int comp) || comp < 0)
        {
            ShowValidationError("Please enter valid number of Competitions.",
txtCompetitions);
            return false;
```

```
        }

        if (!double.TryParse(txtPrivateCoaching.Text, out double hours) || hours < 0)
        {
            ShowValidationError("Please enter valid Private Coaching hours.",
txtPrivateCoaching);
            return false;
        }

        return true;
    }

    // Method 6: Show validation error
    private void ShowValidationError(string message, Control control)
    {
        MessageBox.Show(message, "Validation Error",
            MessageBoxButtons.OK, MessageBoxIcon.Warning);
        control.Focus();
    }

    // FEE CALCULATION METHODS

    // Method 7: Calculate training plan cost
    private double CalculateTrainingPlanCost(int planID)
    {
        try
        {
            string query = "SELECT WeeklyFee FROM TrainingPlans WHERE PlanID = @PlanID";
            SqlParameter[] parameters = {
                new SqlParameter("@PlanID", planID)
            };

            object result = DatabaseHelper.ExecuteScalar(query, parameters);
            double weeklyFee = Convert.ToDouble(result);

            return weeklyFee * 4; // Monthly cost
        }
        catch
        {
            return 0;
        }
    }

    // Method 8: Calculate competition cost
```

```
    private double CalculateCompetitionCost(string trainingPlan, ref int numCompetitions)
    {
        // Beginners cannot compete
        if (trainingPlan == "Beginner" && numCompetitions > 0)
        {
            MessageBox.Show("⚠ Beginner athletes cannot enter competitions!\n\n" +
                "Competition count set to 0.",
                "Information", MessageBoxButtons.OK, MessageBoxIcon.Information);
            numCompetitions = 0;
            txtCompetitions.Text = "0";
        }

        const double COMPETITION_FEE = 220.00;
        return numCompetitions * COMPETITION_FEE;
    }

    // Method 9: Calculate private coaching cost
    private double CalculatePrivateCoachingCost(ref double hours)
    {
        const double MAX_HOURS = 20.0;
        const double HOURLY_RATE = 90.50;

        if (hours > MAX_HOURS)
        {
            MessageBox.Show($"⚠ Maximum private coaching is 5 hours/week
({MAX_HOURS} hours/month).\n\n" +
                $"Hours adjusted to {MAX_HOURS}.",
                "Information", MessageBoxButtons.OK, MessageBoxIcon.Information);
            hours = MAX_HOURS;
            txtPrivateCoaching.Text = MAX_HOURS.ToString();
        }

        return hours * HOURLY_RATE;
    }

    // Method 10: Calculate total monthly cost
    private double CalculateTotalMonthlyCost(double planCost, double compCost, double
coachingCost)
    {
        return planCost + compCost + coachingCost;
    }

    // Method 11: Get weight category upper limit
    private double GetWeightCategoryLimit(int categoryID)
```

```
    {
        try
        {
            string query = "SELECT UpperWeightLimit FROM WeightCategories WHERE
CategoryID = @CategoryID";
            SqlParameter[] parameters = {
        new SqlParameter("@CategoryID", categoryID)
    };

            object result = DatabaseHelper.ExecuteScalar(query, parameters);
            return Convert.ToDouble(result);
        }
        catch
        {
            return 0;
        }
    }

    // Method 12: Compare current weight with category limit
    private string CompareWeightWithCategory(double currentWeight, double
categoryLimit)
    {
        if (categoryLimit > 100)
        {
            if (currentWeight > 100)
                return "Athlete is in correct weight category.";
            else
                return "Athlete can increase weight to match category.";
        }
        else
        {
            if (currentWeight <= categoryLimit)
                return "Athlete is in correct weight category.";
            else
                return "Athlete needs to reduce weight.";
        }
    }


    // Method 13: Display itemized fee breakdown
    private void DisplayFeeBreakdown(string athleteName, string trainingPlan, double
currentWeight,
        string weightCategory, int numCompetitions, double privateCoachingHours)
    {
```

```
        DataTable dt = new DataTable();
        dt.Columns.Add("Item", typeof(string));
        dt.Columns.Add("Details", typeof(string));
        dt.Columns.Add("Amount", typeof(string));

        dt.Rows.Add("═══════════════════════════════",
"═══════════════════════════════", "═══════════════");
        dt.Rows.Add("ATHLETE:", athleteName.ToUpper(), "");
        dt.Rows.Add("MONTH:", dtpMonth.Value.ToString("MMMM yyyy"), "");
        dt.Rows.Add("═══════════════════════════════",
"═══════════════════════════════", "═══════════════");
        dt.Rows.Add("", "", "");

        dt.Rows.Add("ITEM", "DESCRIPTION", "AMOUNT (Rs.)");
        dt.Rows.Add("———————————————", "———————————————",
"———————————");
        dt.Rows.Add("Training Plan", trainingPlan, trainingPlanCost.ToString("N2"));
        dt.Rows.Add("Competitions", $"{numCompetitions} entries",
competitionCost.ToString("N2"));
        dt.Rows.Add("Private Coaching", $"{privateCoachingHours} hours",
privateCoachingCost.ToString("N2"));
        dt.Rows.Add("═══════════════════════════════",
"═══════════════════════════════", "═══════════════");
        dt.Rows.Add("TOTAL MONTHLY COST", "", totalMonthlyCost.ToString("N2"));
        dt.Rows.Add("═══════════════════════════════",
"═══════════════════════════════", "═══════════════");
        dt.Rows.Add("", "", "");

        dt.Rows.Add("WEIGHT ANALYSIS", "", "");
        dt.Rows.Add("———————————————", "———————————————",
"———————————");
        dt.Rows.Add("Current Weight:", $"{currentWeight} kg", "");
        dt.Rows.Add("Category:", weightCategory, "");
        dt.Rows.Add("Status:", weightStatus, "");


        dgvOutput.DataSource = dt;

        dgvOutput.ColumnHeadersVisible = false;
        dgvOutput.RowHeadersVisible = false;
        dgvOutput.AllowUserToResizeRows = false;
        dgvOutput.AllowUserToResizeColumns = false;
        dgvOutput.SelectionMode = DataGridViewSelectionMode.FullRowSelect;
        dgvOutput.MultiSelect = false;
```

```
        dgvOutput.ReadOnly = true;

        dgvOutput.Columns[0].Width = 150;
        dgvOutput.Columns[1].Width = 140;
        dgvOutput.Columns[2].Width = 100;

        dgvOutput.Columns[2].DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
        dgvOutput.Columns[2].DefaultCellStyle.Font = new System.Drawing.Font("Segoe UI",
9, System.Drawing.FontStyle.Bold);

        dgvOutput.Rows[11].DefaultCellStyle.BackColor = System.Drawing.Color.LightGreen;
        dgvOutput.Rows[11].DefaultCellStyle.Font = new System.Drawing.Font("Segoe UI",
10, System.Drawing.FontStyle.Bold);
        dgvOutput.Rows[11].DefaultCellStyle.ForeColor = System.Drawing.Color.DarkGreen;

        dgvOutput.Rows[0].DefaultCellStyle.Font = new System.Drawing.Font("Segoe UI", 9,
System.Drawing.FontStyle.Bold);
        dgvOutput.Rows[1].DefaultCellStyle.BackColor = System.Drawing.Color.LightBlue;
        dgvOutput.Rows[2].DefaultCellStyle.BackColor = System.Drawing.Color.LightBlue;

        dgvOutput.Rows[5].DefaultCellStyle.Font = new System.Drawing.Font("Segoe UI", 9,
System.Drawing.FontStyle.Bold);
        dgvOutput.Rows[14].DefaultCellStyle.Font = new System.Drawing.Font("Segoe UI", 9,
System.Drawing.FontStyle.Bold);
    }

    // SAVE TO DATABASE METHOD


    // SAVE BUTTON
    private void btnSave_Click(object sender, EventArgs e)
    {
        if (totalMonthlyCost == 0)
        {
            MessageBox.Show("Please calculate fees first before saving.",
                "Validation Error", MessageBoxButtons.OK, MessageBoxIcon.Warning);
            return;
        }

        DialogResult result = MessageBox.Show(
            "Are you sure you want to save this monthly registration?",
            "Confirm Save",
            MessageBoxButtons.YesNo,
```

```
            MessageBoxIcon.Question);

        if (result != DialogResult.Yes)
            return;

        if (SaveToDatabase())
        {
            MessageBox.Show("✓ Monthly registration saved successfully!",
                "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);
            ClearAllFields();
        }
    }

    // Method 14: Save registration to database
    private bool SaveToDatabase()
    {
        try
        {
            string query = @"INSERT INTO MonthlyRegistrations
                (AthleteID, PlanID, CategoryID, RegistrationMonth, CurrentWeight,
                 NumCompetitions, PrivateCoachingHours, TrainingPlanCost, CompetitionCost,
                 PrivateCoachingCost, TotalMonthlyCost, WeightStatus, CreatedBy)
            VALUES
                (@AthleteID, @PlanID, @CategoryID, @Month, @Weight,
                 @Competitions, @Coaching, @PlanCost, @CompCost,
                 @CoachingCost, @Total, @Status, @CreatedBy)";

            SqlParameter[] parameters = {
                new SqlParameter("@AthleteID", athleteID),
                new SqlParameter("@PlanID", planID),
                new SqlParameter("@CategoryID", categoryID),
                new SqlParameter("@Month", dtpMonth.Value.Date),
                new SqlParameter("@Weight", double.Parse(txtCurrentWeight.Text)),
                new SqlParameter("@Competitions", int.Parse(txtCompetitions.Text)),
                new SqlParameter("@Coaching", double.Parse(txtPrivateCoaching.Text)),
                new SqlParameter("@PlanCost", trainingPlanCost),
                new SqlParameter("@CompCost", competitionCost),
                new SqlParameter("@CoachingCost", privateCoachingCost),
                new SqlParameter("@Total", totalMonthlyCost),
                new SqlParameter("@Status", weightStatus),
                new SqlParameter("@CreatedBy", frmLogin.LoggedInUser)
            };

            int rowsAffected = DatabaseHelper.ExecuteNonQuery(query, parameters);
```

```
            return rowsAffected > 0;
        }
        catch (Exception ex)
        {
            MessageBox.Show($"Error saving to database:\n\n{ex.Message}",
                "Database Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            return false;
        }
    }

    // CLEAR BUTTON
    private void btnClear_Click(object sender, EventArgs e)
    {
        DialogResult result = MessageBox.Show(
            "Are you sure you want to clear all fields?",
            "Confirm Clear",
            MessageBoxButtons.YesNo,
            MessageBoxIcon.Question);

        if (result == DialogResult.Yes)
        {
            ClearAllFields();
        }
    }

    // Method 15: Clear all fields
    private void ClearAllFields()
    {
        txtAthleteID.Clear();
        txtAthleteName.Clear();
        cmbTrainingPlan.SelectedIndex = -1;
        txtCurrentWeight.Clear();
        cmbWeightCategory.SelectedIndex = -1;
        txtCompetitions.Text = "0";
        txtPrivateCoaching.Text = "0";
        dgvOutput.DataSource = null;

        athleteID = 0;
        trainingPlanCost = 0;
        competitionCost = 0;
        privateCoachingCost = 0;
        totalMonthlyCost = 0;
        weightStatus = "";
```

```
            btnSave.Enabled = false;
            txtAthleteID.Focus();
        }

        // CLOSE BUTTON
        private void btnClose_Click(object sender, EventArgs e)
        {
            this.Close();
        }

        private void txtAthleteID_KeyPress(object sender, KeyPressEventArgs e)
        {
            if (!char.IsDigit(e.KeyChar) && e.KeyChar != (char)Keys.Back)
            {
                e.Handled = true;
            }
        }

        private void txtCurrentWeight_KeyPress(object sender, KeyPressEventArgs e)
        {
            if (!char.IsDigit(e.KeyChar) && e.KeyChar != (char)Keys.Back && e.KeyChar != '.')
            {
                e.Handled = true;
            }
        }

        private void txtCompetitions_KeyPress(object sender, KeyPressEventArgs e)
        {
            if (!char.IsDigit(e.KeyChar) && e.KeyChar != (char)Keys.Back)
            {
                e.Handled = true;
            }
        }

        private void txtPrivateCoaching_KeyPress(object sender, KeyPressEventArgs e)
        {
            if (!char.IsDigit(e.KeyChar) && e.KeyChar != (char)Keys.Back && e.KeyChar != '.')
            {
                e.Handled = true;
            }
        }
    }
}
```

View All Registrations Form GUI

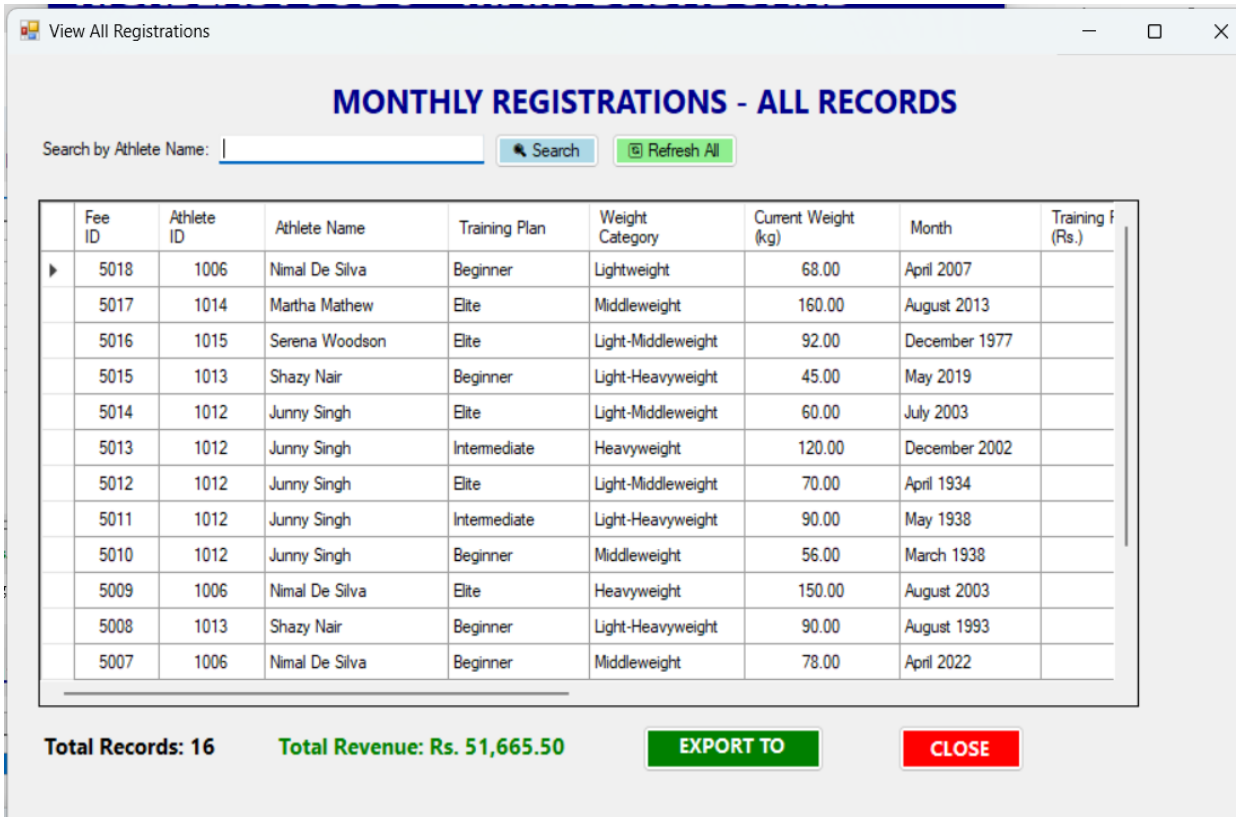


Figure 38: Monthly registrations records (Author developed, 2025)

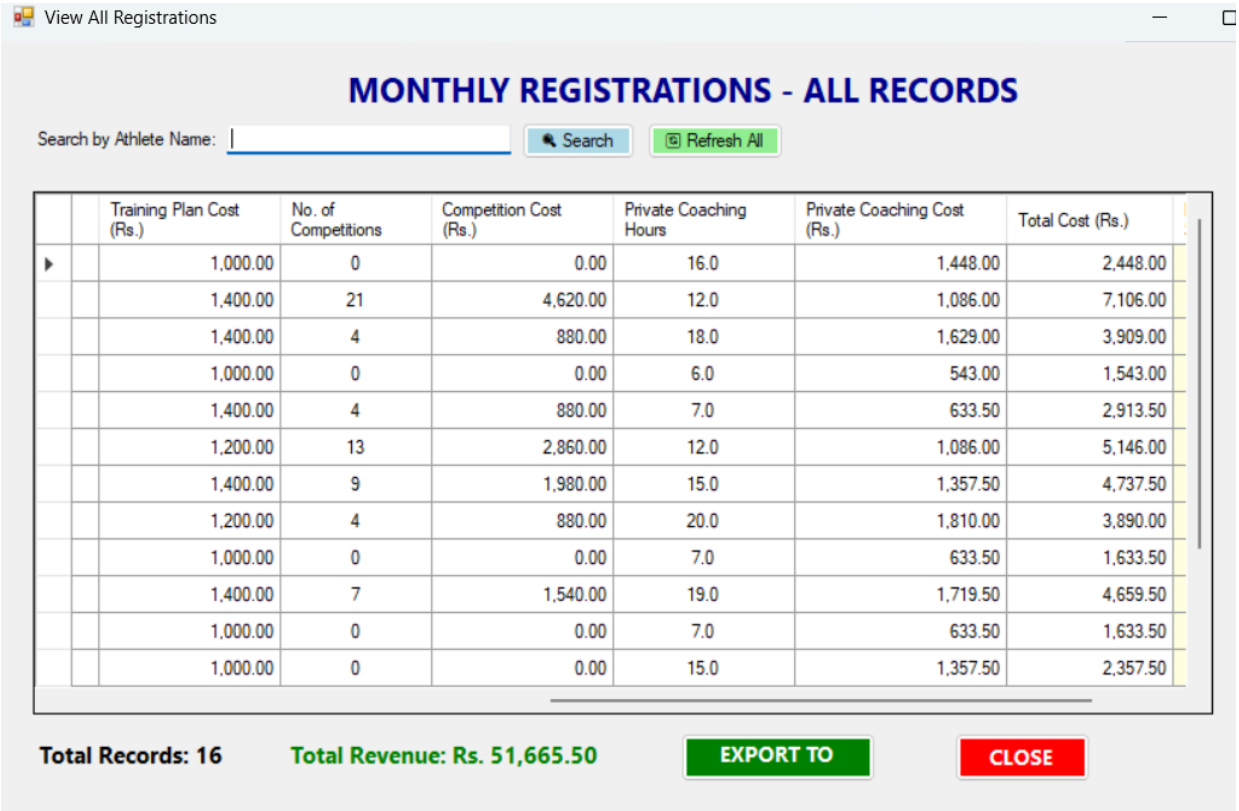


Figure 39: Monthly registration breakdown (Author developed, 2025)

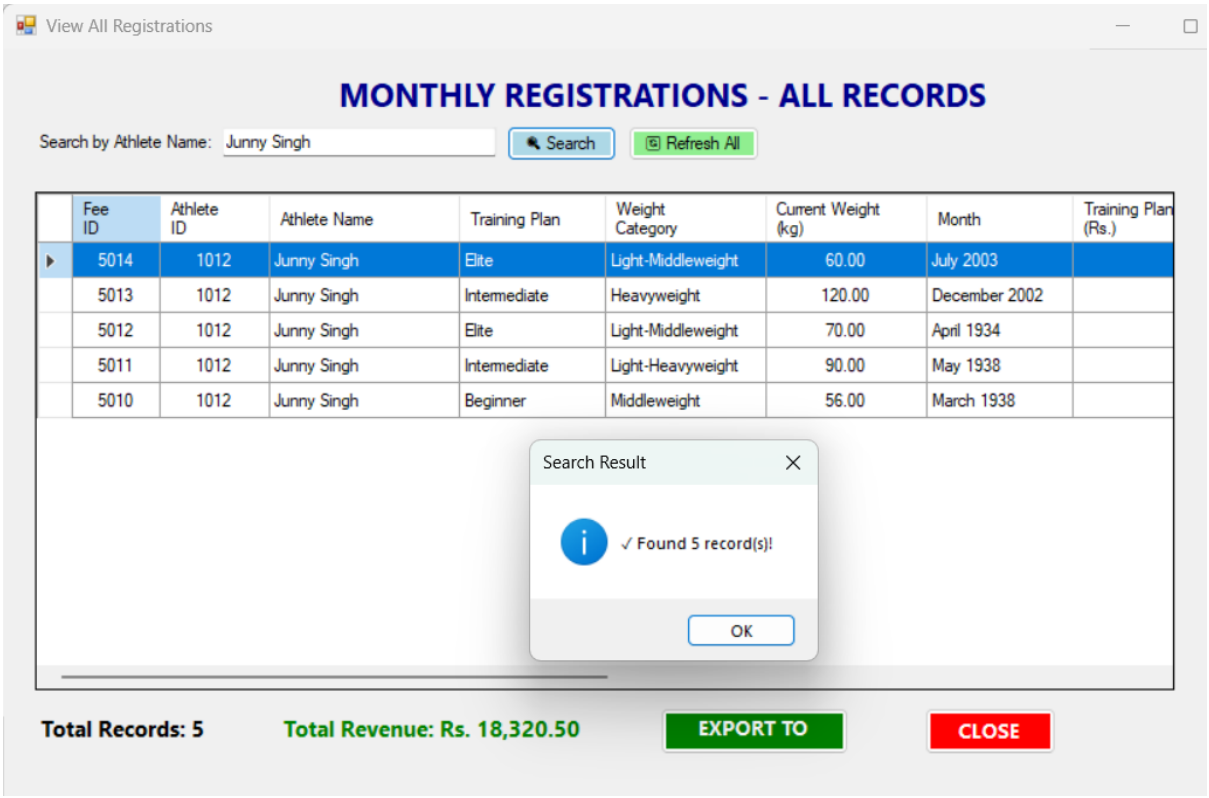


Figure 40: Using the Search button to find Athlete registration details (Author developed, 2025)

View All Registrations Form Code

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Windows.Forms;

namespace KickBlastJudoSystem
{
    public partial class frmViewRegistrations : Form
    {
        public frmViewRegistrations()
        {
            InitializeComponent();
            this.Load += FrmViewRegistrations_Load;
        }

        private void FrmViewRegistrations_Load(object sender, EventArgs e)
        {
            this.StartPosition = FormStartPosition.CenterScreen;
            this.WindowState = FormWindowState.Normal;
```

```
        this.Size = new System.Drawing.Size(900, 600);

        LoadAllRegistrations();
        FormatDataGridView();
    }

    // Method: Load all registrations from database.
    private void LoadAllRegistrations()
    {
        try
        {
            string query = @"
        SELECT
            mr.RegistrationID AS 'Fee ID',
            mr.AthleteID AS 'Athlete ID',
            CONCAT(a.FirstName, ' ', a.LastName) AS 'Athlete Name',
            tp.PlanName AS 'Training Plan',
            wc.CategoryName AS 'Weight Category',
            mr.CurrentWeight AS 'Current Weight (kg)',
            FORMAT(mr.RegistrationMonth, 'MMMM yyyy') AS 'Month',
            mr.TrainingPlanCost AS 'Training Plan Cost (Rs.)',
            mr.NumCompetitions AS 'No. of Competitions',
            mr.CompetitionCost AS 'Competition Cost (Rs.)',
            mr.PrivateCoachingHours AS 'Private Coaching Hours',
            mr.PrivateCoachingCost AS 'Private Coaching Cost (Rs.)',
            mr.TotalMonthlyCost AS 'Total Cost (Rs.)',
            mr.PaymentStatus AS 'Payment Status'
        FROM MonthlyRegistrations mr
        INNER JOIN Athletes a ON mr.AthleteID = a.AthleteID
        INNER JOIN TrainingPlans tp ON mr.PlanID = tp.PlanID
        INNER JOIN WeightCategories wc ON mr.CategoryID = wc.CategoryID
        ORDER BY mr.RegistrationDate DESC";

            DataTable dt = DatabaseHelper.ExecuteQuery(query);
            dgvRegistrations.DataSource = dt;
            UpdateSummary(dt);
        }
        catch (Exception ex)
        {
            MessageBox.Show($"Error loading registrations:\n\n{ex.Message}",
                "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }
```

```
        // Method: DataGridView
        private void FormatDataGridView()
        {
            if (dgvRegistrations.Columns.Count > 0)
            {
                dgvRegistrations.AutoSizeColumnsMode =
DataGridViewAutoSizeColumnsMode.None;
                dgvRegistrations.AllowUserToAddRows = false;
                dgvRegistrations.AllowUserToDeleteRows = false;
                dgvRegistrations.ReadOnly = true;
                dgvRegistrations.SelectionMode = DataGridViewSelectionMode.FullRowSelect;
                dgvRegistrations.MultiSelect = false;
                dgvRegistrations.RowHeadersWidth = 25;


                dgvRegistrations.Columns["Fee ID"].Width = 60;
                dgvRegistrations.Columns["Athlete ID"].Width = 75;
                dgvRegistrations.Columns["Athlete Name"].Width = 130;
                dgvRegistrations.Columns["Training Plan"].Width = 100;
                dgvRegistrations.Columns["Weight Category"].Width = 110;
                dgvRegistrations.Columns["Current Weight (kg)"].Width = 110;
                dgvRegistrations.Columns["Month"].Width = 100;
                dgvRegistrations.Columns["Training Plan Cost (Rs.)"].Width = 120;
                dgvRegistrations.Columns["No. of Competitions"].Width = 100;
                dgvRegistrations.Columns["Competition Cost (Rs.)"].Width = 120;
                dgvRegistrations.Columns["Private Coaching Hours"].Width = 120;
                dgvRegistrations.Columns["Private Coaching Cost (Rs.)"].Width = 140;
                dgvRegistrations.Columns["Total Cost (Rs.)"].Width = 110;
                dgvRegistrations.Columns["Payment Status"].Width = 100;


                dgvRegistrations.Columns["Training Plan Cost (Rs.)"].DefaultCellStyle.Format =
"N2";
                dgvRegistrations.Columns["Competition Cost (Rs.)"].DefaultCellStyle.Format =
"N2";
                dgvRegistrations.Columns["Private Coaching Cost (Rs.)"].DefaultCellStyle.Format =
"N2";
                dgvRegistrations.Columns["Total Cost (Rs.)"].DefaultCellStyle.Format = "N2";


                dgvRegistrations.Columns["Training Plan Cost (Rs.)"].DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
                dgvRegistrations.Columns["Competition Cost (Rs.)"].DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
```

```
            dgvRegistrations.Columns["Private Coaching Cost (Rs.)"].DefaultCellStyle.Alignment
= DataGridViewContentAlignment.MiddleRight;
            dgvRegistrations.Columns["Total Cost (Rs.)"].DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;


            dgvRegistrations.Columns["Fee ID"].DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleCenter;
            dgvRegistrations.Columns["Athlete ID"].DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleCenter;
            dgvRegistrations.Columns["No. of Competitions"].DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleCenter;
            dgvRegistrations.Columns["Private Coaching Hours"].DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleCenter;
            dgvRegistrations.Columns["Payment Status"].DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleCenter;
            dgvRegistrations.Columns["Current Weight (kg)"].DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleCenter;


            foreach (DataGridViewRow row in dgvRegistrations.Rows)
            {
                if (row.Cells["Payment Status"].Value != null)
                {
                    string status = row.Cells["Payment Status"].Value.ToString();
                    if (status == "Paid")
                        row.Cells["Payment Status"].Style.BackColor =
System.Drawing.Color.LightGreen;
                    else if (status == "Pending")
                        row.Cells["Payment Status"].Style.BackColor =
System.Drawing.Color.LightYellow;
                    else if (status == "Overdue")
                        row.Cells["Payment Status"].Style.BackColor =
System.Drawing.Color.LightCoral;
                }
            }
        }
    }

    private void UpdateSummary(DataTable dt)
    {
        int totalRecords = dt.Rows.Count;
        double totalRevenue = 0;

        foreach (DataRow row in dt.Rows)
```

```
        {
            if (row["Total Cost (Rs.)"] != DBNull.Value)
            {
                totalRevenue += Convert.ToDouble(row["Total Cost (Rs.)"]);
            }
        }

        lblTotalRecords.Text = $"Total Records: {totalRecords}";
        lblTotalRevenue.Text = $"Total Revenue: Rs. {totalRevenue:N2}";
    }

    // SEARCH BUTTON
    private void btnSearch_Click(object sender, EventArgs e)
    {
        if (string.IsNullOrWhiteSpace(txtSearch.Text))
        {
            MessageBox.Show("Please enter an athlete name to search.",
                "Validation Error", MessageBoxButtons.OK, MessageBoxIcon.Warning);
            txtSearch.Focus();
            return;
        }

        SearchByAthleteName(txtSearch.Text.Trim());
    }

    // Method: Search registrations by athlete name
    private void SearchByAthleteName(string searchText)
    {
        try
        {
            string query = @"
        SELECT
            mr.RegistrationID AS 'Fee ID',
            mr.AthleteID AS 'Athlete ID',
            CONCAT(a.FirstName, ' ', a.LastName) AS 'Athlete Name',
            tp.PlanName AS 'Training Plan',
            wc.CategoryName AS 'Weight Category',
            mr.CurrentWeight AS 'Current Weight (kg)',
            FORMAT(mr.RegistrationMonth, 'MMMM yyyy') AS 'Month',
            mr.TrainingPlanCost AS 'Training Plan Cost (Rs.)',
            mr.NumCompetitions AS 'No. of Competitions',
            mr.CompetitionCost AS 'Competition Cost (Rs.)',
            mr.PrivateCoachingHours AS 'Private Coaching Hours',
            mr.PrivateCoachingCost AS 'Private Coaching Cost (Rs.)',
```

```
            mr.TotalMonthlyCost AS 'Total Cost (Rs.)',
            mr.PaymentStatus AS 'Payment Status'
        FROM MonthlyRegistrations mr
        INNER JOIN Athletes a ON mr.AthleteID = a.AthleteID
        INNER JOIN TrainingPlans tp ON mr.PlanID = tp.PlanID
        INNER JOIN WeightCategories wc ON mr.CategoryID = wc.CategoryID
        WHERE a.FirstName LIKE @Search
          OR a.LastName LIKE @Search
          OR CONCAT(a.FirstName, ' ', a.LastName) LIKE @Search
          OR CAST(mr.AthleteID AS VARCHAR) LIKE @Search
        ORDER BY mr.RegistrationDate DESC";

            SqlParameter[] parameters = {
                new SqlParameter("@Search", "%" + searchText + "%")
            };

            DataTable dt = DatabaseHelper.ExecuteQuery(query, parameters);

            dgvRegistrations.DataSource = dt;
            FormatDataGridView();
            UpdateSummary(dt);

            if (dt.Rows.Count == 0)
            {
                MessageBox.Show($"No records found for '{searchText}'.\n\nTip: Try searching
by:\n• First name\n• Last name\n• Athlete ID",
                    "Search Result", MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
            else
            {
                MessageBox.Show($"✓ Found {dt.Rows.Count} record(s)!",
                    "Search Result", MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show($"Search error:\n\n{ex.Message}",
                "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }

    // REFRESH BUTTON
    private void btnRefresh_Click(object sender, EventArgs e)
    {
```

```
            txtSearch.Clear();
            LoadAllRegistrations();
            FormatDataGridView();
            MessageBox.Show("✓ Data refreshed successfully!",
                "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }

        // EXPORT BUTTON
        private void btnExport_Click(object sender, EventArgs e)
        {
            MessageBox.Show("Export to Excel feature coming soon!\n\n" +
                "This will export all registration data to an Excel file.",
                "Feature Info", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }

        // CLOSE BUTTON
        private void btnClose_Click(object sender, EventArgs e)
        {
            this.Close();
        }

        private void txtSearch_KeyPress(object sender, KeyPressEventArgs e)
        {
            if (e.KeyChar == (char)Keys.Enter)
            {
                btnSearch_Click(sender, e);
                e.Handled = true;
            }
        }
    }
}
```

Manage Athletes Form GUI

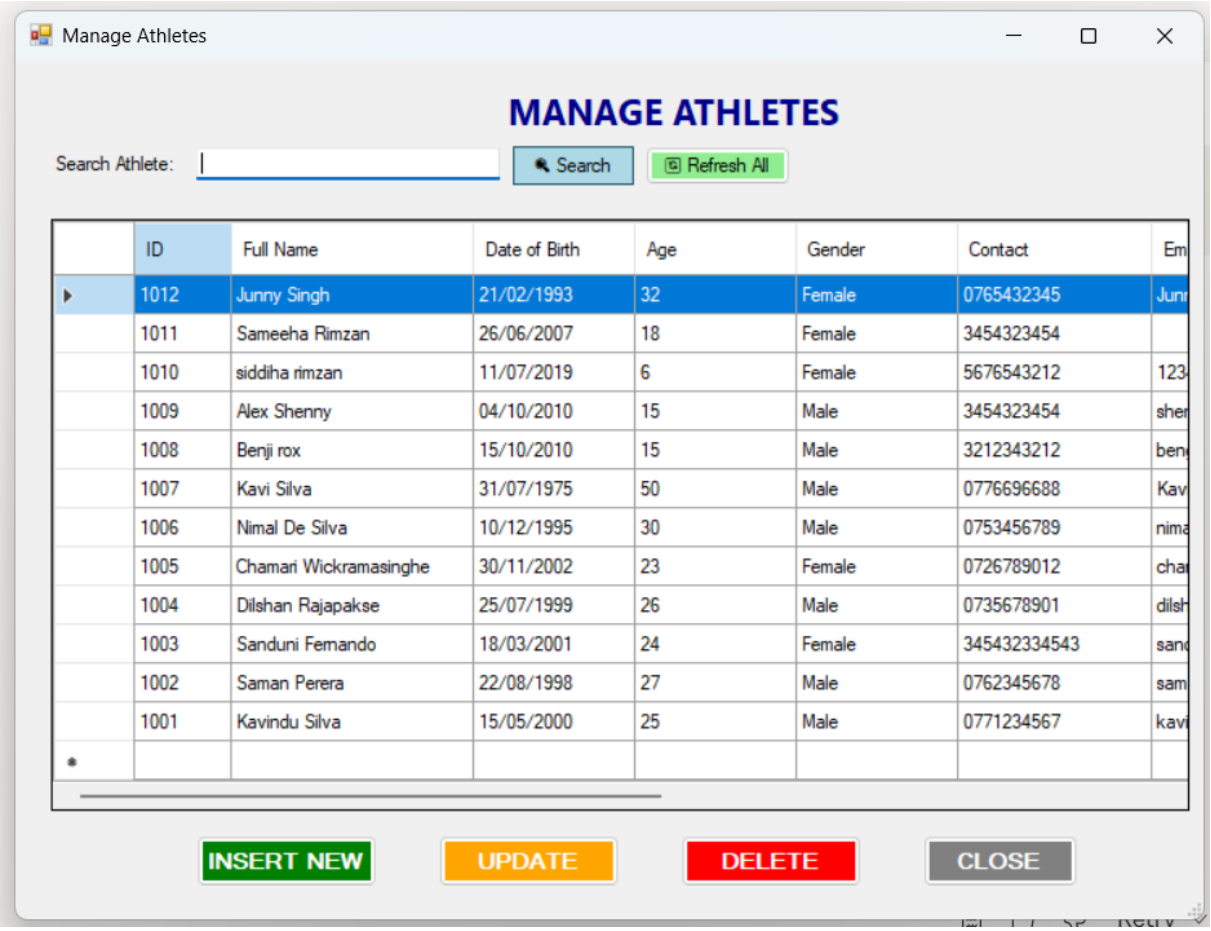


Figure 41: Manage Athletes form (Author developed, 2025)

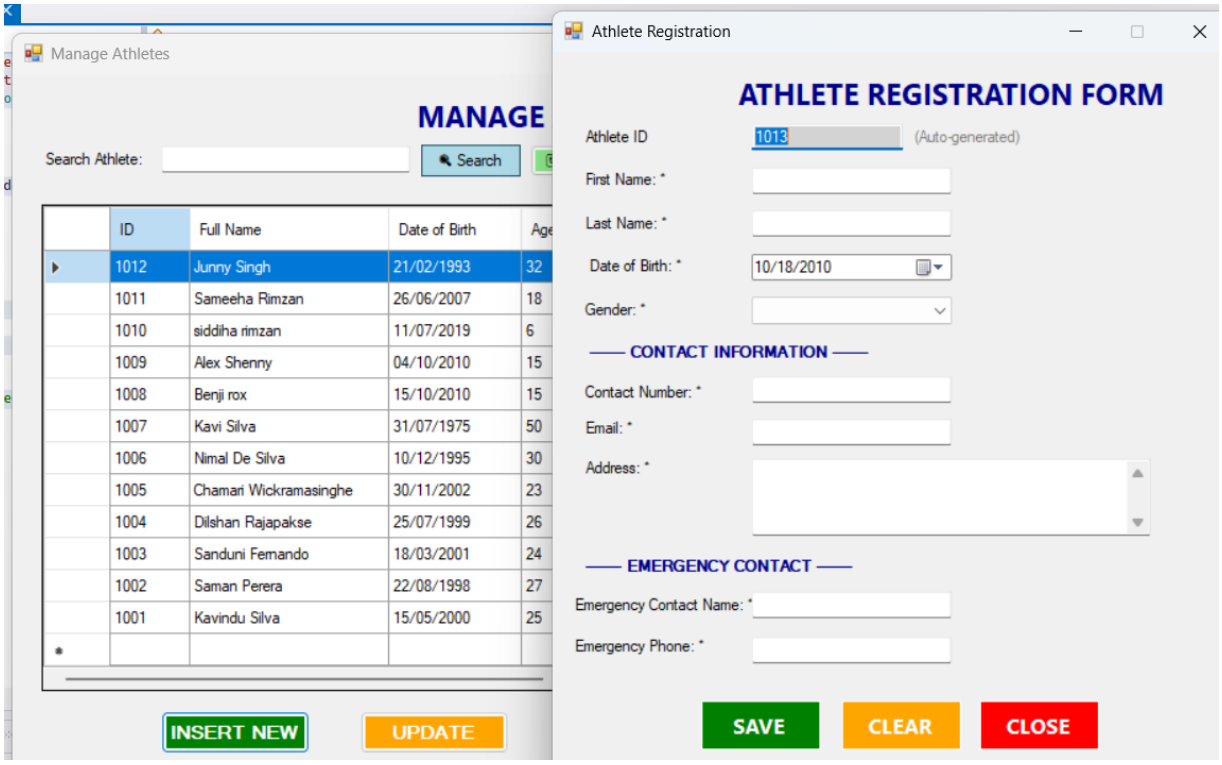


Figure 42: Insert new button adding new Athletes into the system (Author developed, 2025)

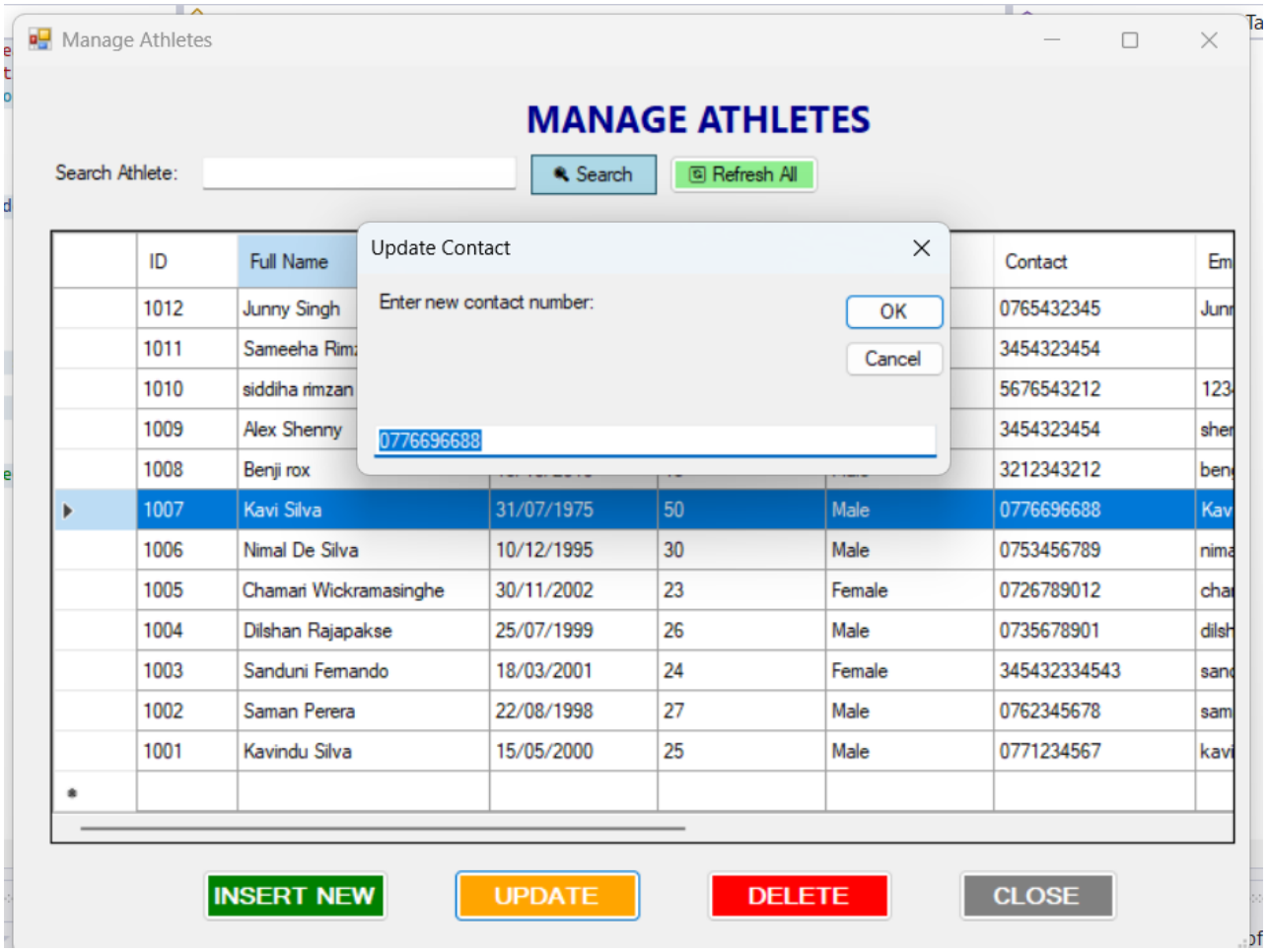


Figure 43: Update button, updating the Contact number (Author developed, 2025)

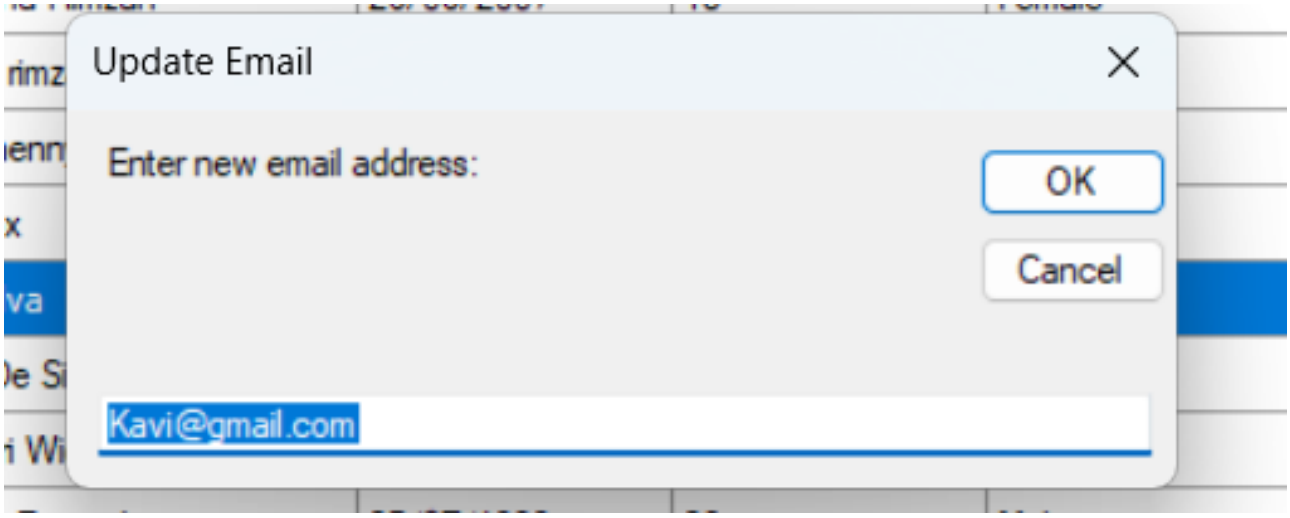


Figure 44: Update button, updating the E-mail address (Author developed, 2025)

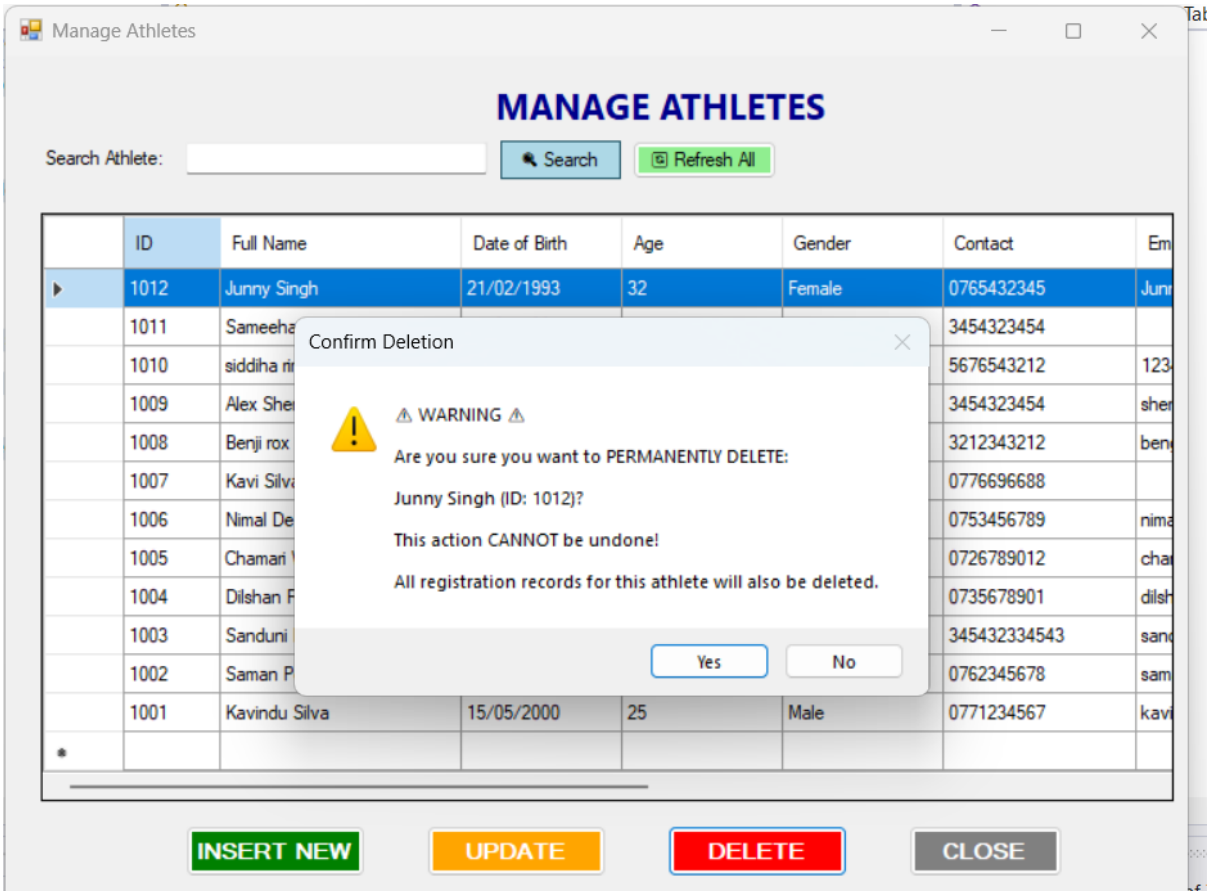


Figure 45: Delete button confirming the deletion of a registration record (Author developed, 2025)

Manage Athletes Form Code

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Windows.Forms;

namespace KickBlastJudoSystem
{
   public partial class frmManageAthletes : Form
   {
      public frmManageAthletes()
      {
         InitializeComponent();
         this.Load += FrmManageAthletes_Load;
      }

      private void FrmManageAthletes_Load(object sender, EventArgs e)
      {
         LoadAllAthletes();
```

```
        FormatGrid();
    }

    private void LoadAllAthletes()
    {
        try
        {
            string query = @"
                SELECT
                    AthleteID AS 'ID',
                    CONCAT(FirstName, ' ', LastName) AS 'Full Name',
                    FORMAT(DateOfBirth, 'dd/MM/yyyy') AS 'Date of Birth',
                    Age,
                    Gender,
                    ContactNumber AS 'Contact',
                    Email,
                    Address,
                    EmergencyContactName AS 'Emergency Contact',
                    EmergencyContactPhone AS 'Emergency Phone',
                    FORMAT(EnrollmentDate, 'dd/MM/yyyy') AS 'Enrolled On',
                    CASE WHEN IsActive = 1 THEN 'Active' ELSE 'Inactive' END AS 'Status'
                FROM Athletes
                ORDER BY AthleteID DESC";

            DataTable dt = DatabaseHelper.ExecuteQuery(query);
            dgvAthletes.DataSource = dt;
            FormatGrid();
        }
        catch (Exception ex)
        {
            MessageBox.Show($"Error loading athletes: {ex.Message}",
                "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }

    private void FormatGrid()
    {
        if (dgvAthletes.Columns.Count > 0)
        {
            dgvAthletes.Columns["ID"].Width = 60;
            dgvAthletes.Columns["Full Name"].Width = 150;
            dgvAthletes.Columns["Contact"].Width = 120;
            dgvAthletes.Columns["Status"].Width = 80;
```

```
            foreach (DataGridViewRow row in dgvAthletes.Rows)
            {
                if (row.Cells["Status"].Value != null)
                {
                    if (row.Cells["Status"].Value.ToString() == "Active")
                        row.Cells["Status"].Style.BackColor = System.Drawing.Color.LightGreen;
                    else
                        row.Cells["Status"].Style.BackColor = System.Drawing.Color.LightCoral;
                }
            }
        }
    }

    // INSERT BUTTON
    private void btnInsert_Click(object sender, EventArgs e)
    {
        frmAthleteRegistration regForm = new frmAthleteRegistration();
        if (regForm.ShowDialog() == DialogResult.OK)
        {
            LoadAllAthletes();
            MessageBox.Show("✓ New athlete added successfully!",
                "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
    }

    // UPDATE BUTTON
    private void btnUpdate_Click(object sender, EventArgs e)
    {
        if (dgvAthletes.SelectedRows.Count == 0)
        {
            MessageBox.Show("⚠ Please select an athlete to update.",
                "Selection Required", MessageBoxButtons.OK, MessageBoxIcon.Warning);
            return;
        }

        int athleteID = Convert.ToInt32(dgvAthletes.SelectedRows[0].Cells["ID"].Value);
        string name = dgvAthletes.SelectedRows[0].Cells["Full Name"].Value.ToString();


        DialogResult result = MessageBox.Show(
            $"Update athlete information for:\n\n{name} (ID: {athleteID})\n\n" +
            "Enter new contact number:",
            "Update Athlete",
            MessageBoxButtons.OKCancel,
```

```
            MessageBoxIcon.Question);

        if (result == DialogResult.OK)
        {
            UpdateAthleteContact(athleteID);
        }
    }

    private void UpdateAthleteContact(int athleteID)
    {
        try
        {
            string query = "SELECT ContactNumber, Email FROM Athletes WHERE AthleteID =
@ID";
            SqlParameter[] parameters = { new SqlParameter("@ID", athleteID) };
            DataTable dt = DatabaseHelper.ExecuteQuery(query, parameters);

            if (dt.Rows.Count > 0)
            {
                string currentContact = dt.Rows[0]["ContactNumber"].ToString();
                string currentEmail = dt.Rows[0]["Email"].ToString();

                // Get new contact number
                string newContact = Microsoft.VisualBasic.Interaction.InputBox(
                    "Enter new contact number:",
                    "Update Contact",
                    currentContact);

                if (string.IsNullOrEmpty(newContact))
                    return;

                // Get new email
                string newEmail = Microsoft.VisualBasic.Interaction.InputBox(
                    "Enter new email address:",
                    "Update Email",
                    currentEmail);

                // Update both contact and email
                string updateQuery = @"UPDATE Athletes
            SET ContactNumber = @Contact,
                Email = @Email,
                ModifiedDate = GETDATE()
            WHERE AthleteID = @ID";
```

```
            SqlParameter[] updateParams = {
                new SqlParameter("@Contact", newContact),
                new SqlParameter("@Email", string.IsNullOrWhiteSpace(newEmail) ?
                    (object)DBNull.Value : newEmail),
                new SqlParameter("@ID", athleteID)
            };

            int rows = DatabaseHelper.ExecuteNonQuery(updateQuery, updateParams);

            if (rows > 0)
            {
                MessageBox.Show("✓ Athlete information updated successfully!\n\n" +
                    $"New Contact: {newContact}\n" +
                    $"New Email: {(string.IsNullOrWhiteSpace(newEmail) ? "Not provided" :
newEmail)}",
                    "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);
                LoadAllAthletes();
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show($"Error updating athlete: {ex.Message}",
                "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }

    // DELETE BUTTON
    private void btnDelete_Click(object sender, EventArgs e)
    {
        if (dgvAthletes.SelectedRows.Count == 0)
        {
            MessageBox.Show("⚠ Please select an athlete to delete.",
                "Selection Required", MessageBoxButtons.OK, MessageBoxIcon.Warning);
            return;
        }

        int athleteID = Convert.ToInt32(dgvAthletes.SelectedRows[0].Cells["ID"].Value);
        string name = dgvAthletes.SelectedRows[0].Cells["Full Name"].Value.ToString();

        DialogResult result = MessageBox.Show(
            $"⚠ WARNING ⚠\n\nAre you sure you want to PERMANENTLY DELETE:\n\n" +
            $"{name} (ID: {athleteID})?\n\n" +
            "This action CANNOT be undone!\n\n" +
```

```
            "All registration records for this athlete will also be deleted.",
            "Confirm Deletion",
            MessageBoxButtons.YesNo,
            MessageBoxIcon.Warning);

        if (result == DialogResult.Yes)
        {
            DeleteAthlete(athleteID);
        }
    }

    private void DeleteAthlete(int athleteID)
    {
        try
        {
            string query = "DELETE FROM Athletes WHERE AthleteID = @ID";
            SqlParameter[] parameters = {
                new SqlParameter("@ID", athleteID)
            };

            int rowsAffected = DatabaseHelper.ExecuteNonQuery(query, parameters);

            if (rowsAffected > 0)
            {
                MessageBox.Show("✓ Athlete deleted successfully!\n\n" +
                    "All related records have been removed from the database.",
                    "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);
                LoadAllAthletes();
            }
            else
            {
                MessageBox.Show("✗ Failed to delete athlete.",
                    "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show($"Error deleting athlete:\n\n{ex.Message}\n\n" +
                "Note: Cannot delete athlete with existing registrations.\n" +
                "Delete their registrations first, or use 'Deactivate' instead.",
                "Database Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }
```

```
// SEARCH BUTTON
private void btnSearch_Click(object sender, EventArgs e)
{
    if (string.IsNullOrWhiteSpace(txtSearch.Text))
    {
        MessageBox.Show("Please enter a search term.", "Validation",
            MessageBoxButtons.OK, MessageBoxIcon.Warning);
        return;
    }

    try
    {
        string query = @"
            SELECT
                AthleteID AS 'ID',
                CONCAT(FirstName, ' ', LastName) AS 'Full Name',
                FORMAT(DateOfBirth, 'dd/MM/yyyy') AS 'Date of Birth',
                Age,
                Gender,
                ContactNumber AS 'Contact',
                Email,
                Address,
                EmergencyContactName AS 'Emergency Contact',
                EmergencyContactPhone AS 'Emergency Phone',
                FORMAT(EnrollmentDate, 'dd/MM/yyyy') AS 'Enrolled On',
                CASE WHEN IsActive = 1 THEN 'Active' ELSE 'Inactive' END AS 'Status'
            FROM Athletes
            WHERE FirstName LIKE @Search OR LastName LIKE @Search
                OR CAST(AthleteID AS VARCHAR) LIKE @Search
            ORDER BY AthleteID DESC";

        SqlParameter[] parameters = {
            new SqlParameter("@Search", "%" + txtSearch.Text.Trim() + "%")
        };

        DataTable dt = DatabaseHelper.ExecuteQuery(query, parameters);
        dgvAthletes.DataSource = dt;
        FormatGrid();

        if (dt.Rows.Count == 0)
        {
            MessageBox.Show("No athletes found.", "Search Result",
                MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
```

```
        }
        catch (Exception ex)
        {
            MessageBox.Show($"Search error: {ex.Message}",
                "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }

    // REFRESH BUTTON
    private void btnRefresh_Click(object sender, EventArgs e)
    {
        txtSearch.Clear();
        LoadAllAthletes();
        MessageBox.Show("✓ Data refreshed!", "Success",
            MessageBoxButtons.OK, MessageBoxIcon.Information);
    }

    // CLOSE BUTTON
    private void btnClose_Click(object sender, EventArgs e)
    {
        this.Close();
    }

    private void txtSearch_KeyPress(object sender, KeyPressEventArgs e)
    {
        if (e.KeyChar == (char)Keys.Enter)
        {
            btnSearch_Click(sender, e);
        }
    }
  }
}
```

Database Table Design

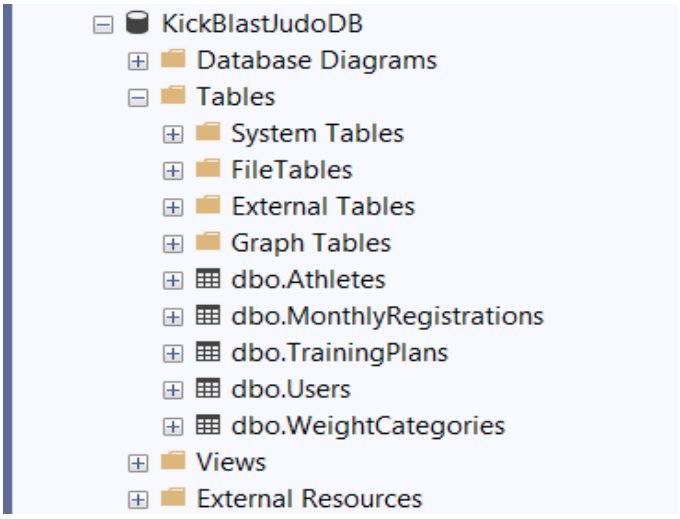


Figure 46: SQL database tables created for the KickBlast Judo System (Author developed, 2025)

DELL-LAPTOP...dbo.Athletes*

Column Name	Data Type	Allow Nulls
AthleteID	int	☐
FirstName	varchar(50)	☐
LastName	varchar(50)	☐
DateOfBirth	date	☐
Age		☑
Gender	varchar(10)	☐
ContactNumber	varchar(15)	☐
Email	varchar(100)	☑
Address	varchar(255)	☑
EmergencyContactName	varchar(100)	☐
EmergencyContactPhone	varchar(15)	☐
EnrollmentDate	date	☑
IsActive	bit	☑
CreatedBy	varchar(50)	☑
CreatedDate	datetime	☑
ModifiedDate	datetime	☑

Figure 47: Design view of the Athletes table (Author developed, 2025)

DELL-LAPTOP\...htCategories | DELL-LAPTOP\S...B - dbo.Users

Column Name	Data Type	Allow Nulls
CategoryID	int	☐
CategoryName	varchar(50)	☐
UpperWeightLimit	decimal(5, 2)	☐
GenderApplicable	varchar(10)	☑
IsActive	bit	☑
CreatedDate	datetime	☑

Figure 48: Design view of Weight Categories table (Author developed, 2025)

DELL-LAPTOP\...rainingPlans

Column Name	Data Type	Allow Nulls
PlanID	int	☐
PlanName	varchar(50)	☐
SessionsPerWeek	int	☐
WeeklyFee	decimal(10, 2)	☐
MonthlyFee		☑
CanCompete	bit	☐
Description	varchar(255)	☑
IsActive	bit	☑
CreatedDate	datetime	☑
		☐

Figure 49: Figure 23: Design view of Training Plans table (Author developed, 2025)

DELL-LAPTOP\...egistrations

Column Name	Data Type	Allow Nulls
RegistrationID	int	☐
AthleteID	int	☐
PlanID	int	☐
CategoryID	int	☐
RegistrationMonth	date	☐
CurrentWeight	decimal(5, 2)	☐
NumCompetitions	int	☑
PrivateCoachingHours	decimal(4, 1)	☑
TrainingPlanCost	decimal(10, 2)	☐
CompetitionCost	decimal(10, 2)	☑
PrivateCoachingCost	decimal(10, 2)	☑
TotalMonthlyCost	decimal(10, 2)	☐
WeightStatus	varchar(100)	☑
PaymentStatus	varchar(20)	☑
Notes	varchar(500)	☑
CreatedBy	varchar(50)	☑
RegistrationDate	datetime	☑
ModifiedDate	datetime	☑

Figure 50: Design view of Monthly Registrations table (Author developed, 2025)

DELL-LAPTOP\... - dbo.Users

Column Name	Data Type	Allow Nulls
UserID	int	☐
Username	varchar(50)	☐
Password	varchar(255)	☐
FullName	varchar(100)	☐
Role	varchar(20)	☑
IsActive	bit	☑
CreatedDate	datetime	☑
LastLogin	datetime	☑

Figure 51: Design view of the Users table (Author developed, 2025)

4.2 Introduction to Debugging

Debugging

Debugging is the process of identifying, analyzing, and fixing errors or unexpected behaviors in a program (Techopedia, 2024). It ensures to check the flow of the application, show the exact point where the problem occurred and make the required adjustments to fix the error so that the program can run as intended. In Visual Studio, debugging can be done using tools such as breakpoints, step-through execution and variable inspection, which help developers trace and fix issues efficiently.

Importance of Debugging

Debugging is essential because it improves the overall reliability, functionality and quality of a program by carefully checking the code during execution. Moreover, developers can detect both syntax and logic errors that may not be immediately visible during development. This helps to prevent crashes, incorrect calculations and security vulnerabilities in the system. An effective debugging ensures that the final application will function correctly, provide a better user experience and reduces the risk of future maintenance issues.

Features available in Visual studio IDE for debugging:

Login Form Interface:

The login form interface was designed to allow users to securely and safely access the KickBlast Judo System. It includes input fields for entering the username and password with Login and Exit buttons. When the user clicks on the login button, the application initiates the authentication process by checking the entered credentials, this interface acts as the entry point to the system and is needed for testing the functionality and debugging the login process during development.

Figure 52: Login form interface of KickBlast Judo system (Author developed, 2025)

Login Button Click Event Code Implementation

```
private void btnLogin_Click(object sender, EventArgs e)
{
    // Step 1: Validate inputs
    if (!ValidateInputs())
        return;

    // Step 2: Authenticate user
    string username = txtUsername.Text.Trim();
    string password = txtPassword.Text;

    if (AuthenticateUser(username, password))
    {
        // Step 3: Update last login
        UpdateLastLogin(username);

        // Step 4: Store logged-in user
        LoggedInUser = username;

        // Step 5: Show success message
        MessageBox.Show($"✓ Welcome to KickBlast Judo System!\n\nLogged in as: {username}",
            "Login Successful", MessageBoxButtons.OK, MessageBoxIcon.Information);

        // Step 6: Open Main Dashboard (TEMPORARILY COMMENTED - Will enable after creating dashboard)

        this.Hide();
        frmMainDashboard dashboard = new frmMainDashboard();
        dashboard.ShowDialog();

        // Step 7: When dashboard closes, show login again
        this.Show();
        ClearFields();
        LoggedInUser = "";


        // TEMPORARY - Just clear fields for testing
        ClearFields();
        MessageBox.Show("Dashboard will open here after we create it!",
            "Info", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    else
    {
        MessageBox.Show("✗ Invalid username or password!\n\nPlease try again.",
            "Login Failed", MessageBoxButtons.OK, MessageBoxIcon.Error);
        txtPassword.Clear();
        txtPassword.Focus();
    }
}
```

Figure 53: Login event code implementation for KickBlast Judo System (Author developed, 2025)

This image shows the implementation of the btnLogin_Click event in the KickBlast Judo system. When a user clicks the "Login" button, this method is responsible for handling the entire login process and the code is easier to follow and maintain because it is divided into clear steps.

- **Step 1** checks the username and password inputs by calling the ValidateInputs() method.

- **Step 2** calls the AuthenticateUser function to check the entered credentials against the database.
- **Steps 3 & 4** In order to ensure that the user activity is correctly sent to the system these steps update the most recent login information and store the currently loggin-in username.
- **Step 5** displays a customized welcome message for the logged-in user.
- **Step 6** contains the code to open the main dashboard form, which is temporarily commented out until the dashboard interface is fully developed.
- **Step 7** ensures that when the dashboard is closed, the login form reappears, and fields are cleared.

By structuring the code this way and calling separate methods for each operation, makes the login process becomes readable and easy to debug which satisfies the KickBlast Judo organization's need for a system that is both professional and maintainable.

Breakpoint Set on Login Button Event

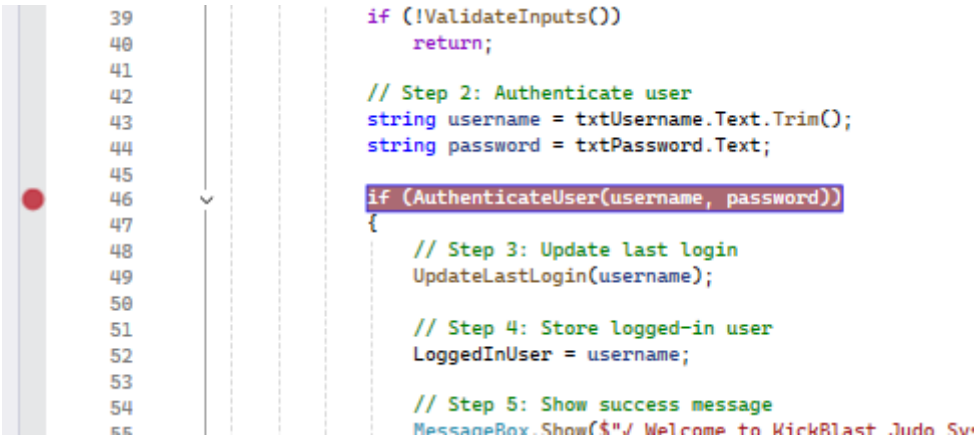


Figure 54: Breakpoint set at the start of the login method (Author developed, 2025)

In this figure, a breakpoint is placed at the start of the btnLogin_Click method. This allows the program execution to pause when the login button is clicked. Using this debugging technique helped me carefully trace the flow of the login process step by step and ensure that each method was being called in the correct order.

For the KickBlast Judo System, this was particularly helpful to check that the validation and authentication steps were functioning correctly before accessing the dashboard.

Debugging in Progress

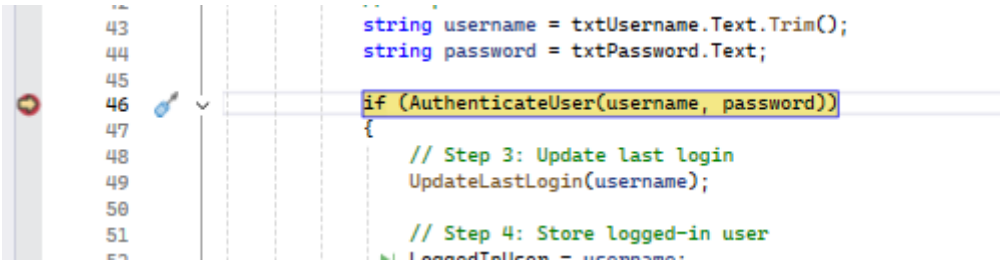


Figure 55: Debugging process using yellow execution line (Author developed, 2025)

This screenshot shows that the program has paused at the breakpoint and is ready to step through the code by using "Step over" and "Step Into" options in Visual Studio, I was able to inspect variables values like entered username and password, watch the flow between different methods and quickly identify possible logical error. This process ensures that incorrect credentials were properly handled and that the correct success message appeared only for valid logins.

Database Connection Error Displayed in Output Window

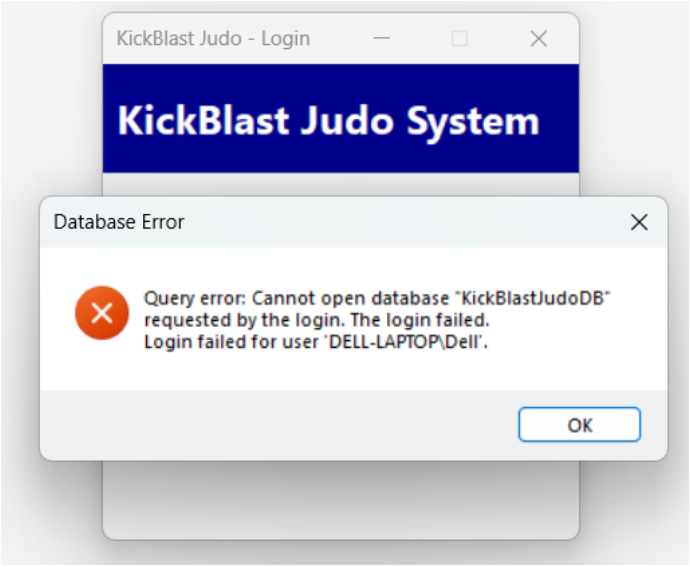


Figure 56: Database connection error traced through the Visual Studio Output Window during debugging (Author developed, 2025)

This shows a database related error encountered during the login process of KickBlast Judo system, when the application attempted to a authenticate a user, an error message appeared in the Visual Studio Output Window, showing that the database connection was not properly established and connected to the system.

Since this message indicated where the issue was, I was able to quickly check that the problem was related to the database setup and connection string rather than the GUI or the login logic. Therefore, it allowed me to trace the exact cause of the issue and plan the next steps to configure and connect the database correctly. This demonstrates how the Output Window is essential to debugging because it helps developers identify and pinpoint issues in real-time making it very efficient and effective.

Step Over, Step Into, Step Out

These allow you to control exactly how the code is executed line by line

Figure 57: Step Over, Step Into and Step Out (Pawal Grzybek, 2021)

- **Step Over (F10)** – executes the current line and moves to the next without entering functions.

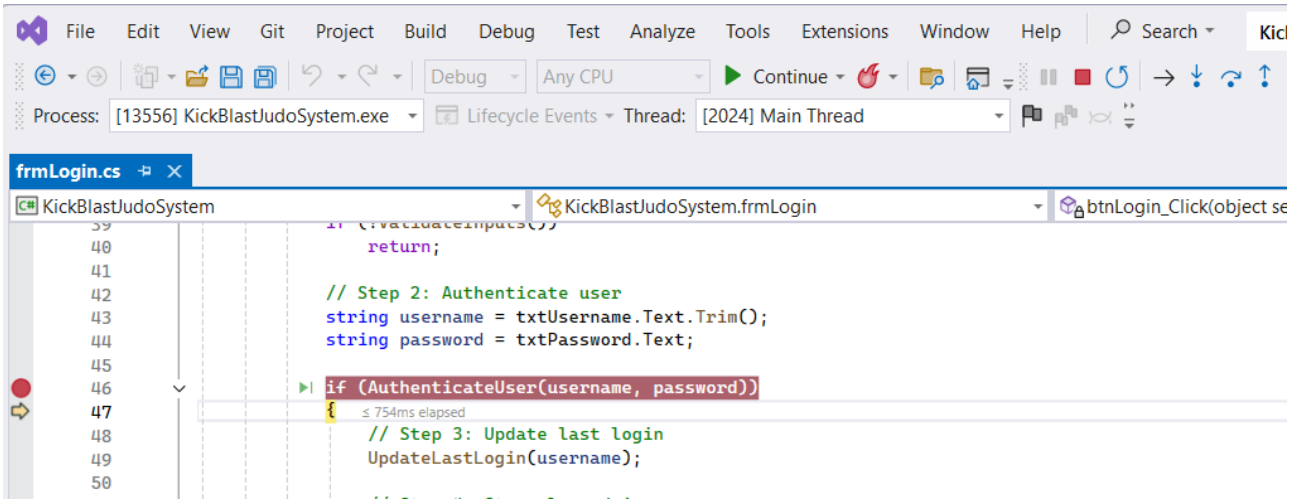


Figure 58: Using Step Over in Visual Studio to move through the login code line by line without entering each method (Author developed, 2025).

- **Step Into (F11)** – goes inside the called method, useful to debug functions like AuthenticateUser().

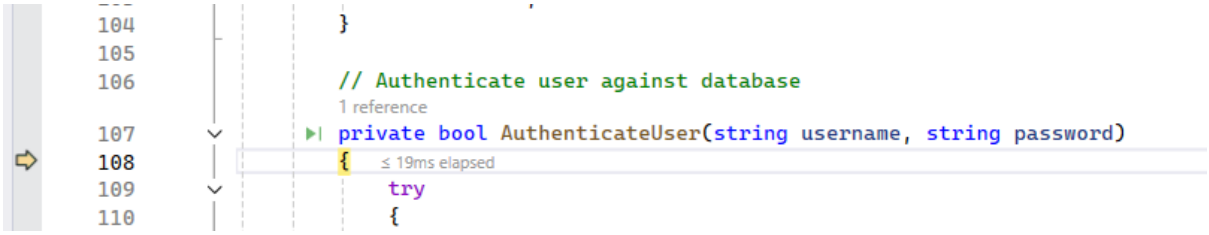


Figure 59: Using Step Into to trace the flow inside the AuthenticateUser method for detailed debugging (Author developed, 2025).

- **Step Out (Shift + F11)** – quickly returns from a method to the caller.

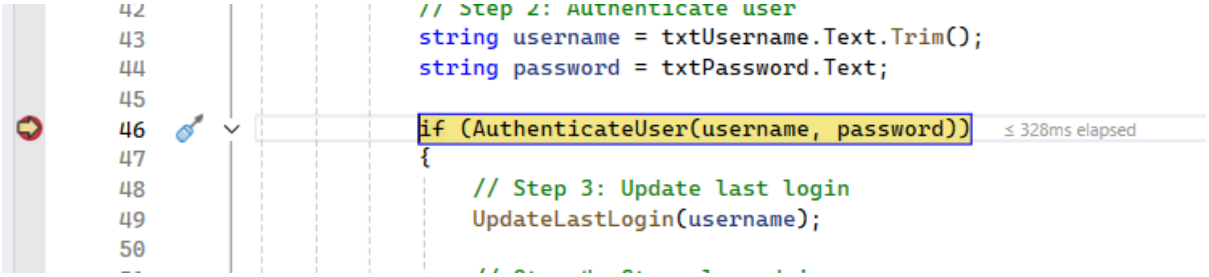


Figure 60: Using Step Out to return from the AuthenticateUser method back to the calling method in the login flow (Author developed, 2025).

This helped me follow the login flow clearly and check that each method in the authentication process worked as intended.

Quick Watch

Quick Watch allows you to check the current value of variables or expressions while the program is paused at a breakpoint and it is great for examining whether the right data is being passed into functions, for example, verifying the username value during the login process of the KickBlast Judo System.

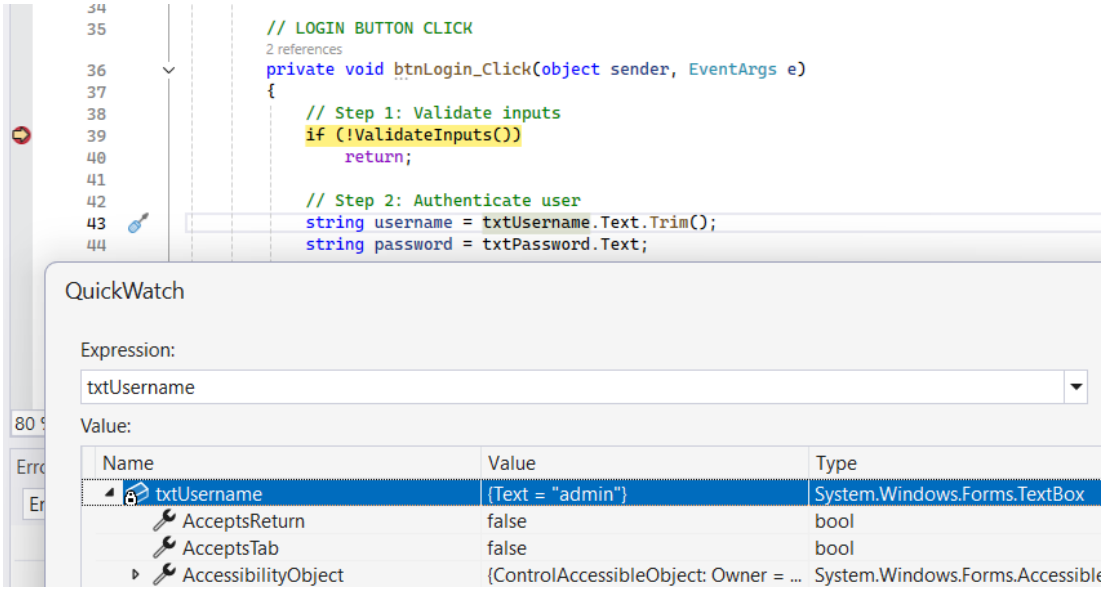


Figure 61: Using Quick Watch to check the value of the username variable during the login process (Author developed, 2025).

During debugging, I used Quick Watch to check the values stored in variables like username before they were passed into the AuthenticateUser() method. This helped me confirm that the user input was being correctly captured and sent to the database, making it easier to identify logical issues early in the process.

Add Watch

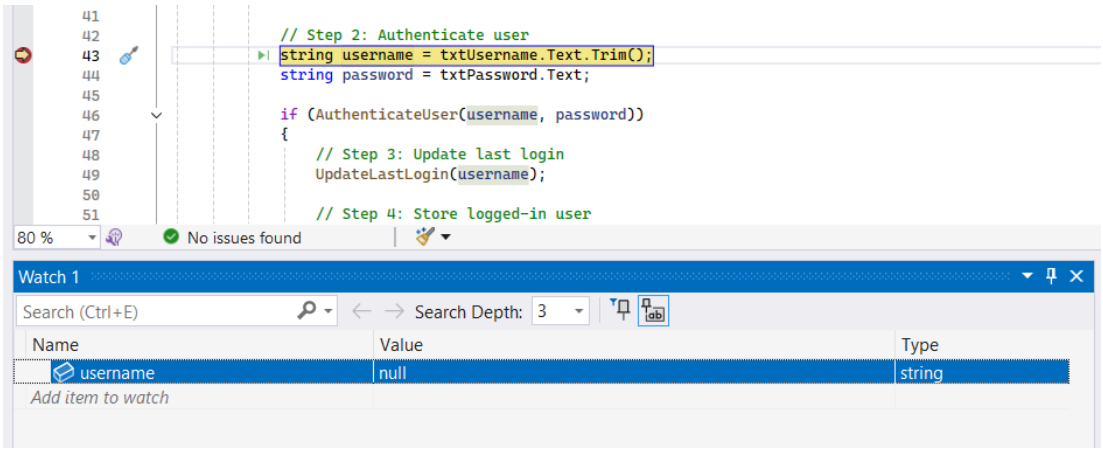


Figure 62: Using Add Watch in Visual Studio to continuously monitor the value of the username variable during debugging (Author developed, 2025).

The Add Watch feature allows me to continuously check the values of selected variables while the program is running in Debug mode. In this example, I added the username variable to the Watch window during the login process of the KickBlast Judo System. This enabled

me to observe how the value changed in real time and verify that the input data was being captured and passed correctly throughout the authentication flow and this is especially useful for tracking variables across multiple breakpoints, helping to ensure that the application is more secure and robust by catching unexpected or null values early.

Evaluation of Debugging in Developing a Secure and Robust Application

The process of debugging was essential in the development of KickBlast Judo system, helping me ensure that the final application was both secure and robust. Throughout the development process, a structured debugging approach was used to track data flow, analyze system behavior and detect vulnerabilities early rather than just to fix silly errors. Moreover, by effectively using built-in debugging tools which was available in Visual Studio IDE, it was easy to strength the overall quality of the system, making it more secure and stable.

From the very beginning of the project, debugging was not just used only when errors occurred but rather developed a habit to detect issues early. For example, during the development of the login authentication feature, I purposely set breakpoints at critical points such as the start of the btnLogin_Click event, this allowed me to pause the execution before any sensitive information like usernames and passwords was processed. By stepping through the code line by using Step Into and Step Over, I could clearly see the exact order in which validation checks and authentication methods were triggered. This was extremely useful because it allowed me to catch issues early such as missing input validation or incorrect method calls, as both could have led to serious security flaws like bypassing login authentication or unintentionally allowing blank inputs.

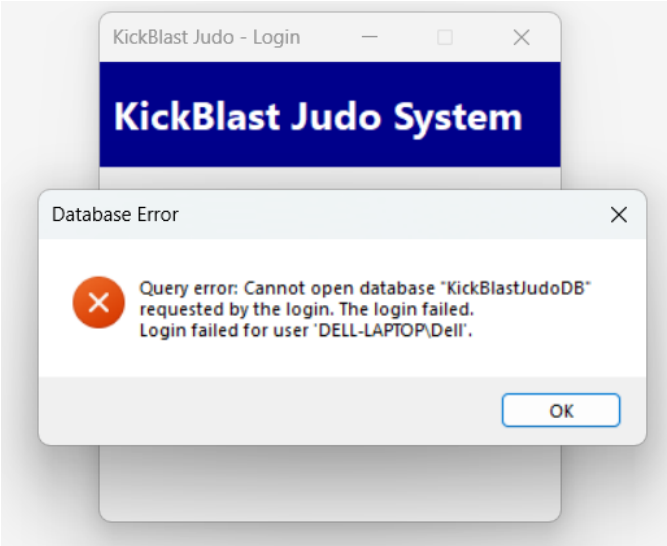


Figure 63: Database connection error (Author developed, 2025)

One of the most impactful debugging moments was when I encountered a database connection error during testing shown in the figure above. Instead of assuming the problem was with the user interface, I carefully observed the Output Window, which displayed a detailed error message indicating that the issue was linked to the database. This allowed me to check the connection string in the code and the database server settings, after making the necessary adjustments, the error was resolved. This experience showed me how debugging is more than just code and how it can help to identify and fix deeper issues, which is essential for ensuring that the system runs smoothly.

In order to track the flow of data in real-time, Quick watch and Add watch features were also used. For instance, while debugging the login form, I added watches to variables like txtUsername.Text and txtPassword.Text to make sure that the values entered by the user were correctly passed into the authentication methods, this helped me detect and fix unexpected behaviors such as null values being passed through as well. Therefore, by continuously checking these variables, I could make sure that the inputs were accurate and secure which could prevent issues like blank credentials.

Debugging also improved the performance of the application in form navigation and data handling across many components. The KickBlast Judo System involved various interconnected forms such as registration, login, and training fee calculation. Using breakpoints and Step Into allowed me to trace how data moved between these forms, and certain variables were not being updated properly which could have caused errors if they were not corrected. Debugging helped me pinpoint these logic errors quickly and implement right fixes to maintain consistency throughout the system.

Not only that, debugging helped in improving the overall design and structure of the code. By closely checking the logic regularly, I noticed and modified repetitive code, add better data handling and more meaning validation.

For example, while debugging the training fee calculation, I realized that some input checks were missing, which could have allowed invalid data to enter the system, after identifying these gaps, I added clear validation rules and appropriate exception handling, which made the system more reliable and helped to reduce unexpected crashes.

Furthermore, debugging improved my own development skills. Before this project, I used to view debugging as something to do at the end of the system development. However, working on the KickBlast Judo system altered that. I learned to integrate debugging throughout the development lifecycle, which not only reduced the number of unexpected errors but also made me more confident in the code I was writing.

In conclusion, debugging was not just a tool but a structured development strategy that helped to build a secure, stable and fully functional application. Through the use of breakpoints, step controls, Quick Watch, Add Watch, and the Output Window, I was able to carefully analyse the behaviour of the system at every stage and fix issues effectively and efficiently. Additionally, Debugging helped me strengthen the authentication process, improve data handling, resolve database connectivity problems, and correct the program structure. Overall, debugging increased the quality of the application and enhanced my skills as a developer, proving to be one of the most valuable practices in the entire project.

4.3 Coding Standards

Explanation of Coding Standards

Coding Standards are a set of rules and guidelines that define how code should be written, formatted and structured (Coding Ninjas, 2023). These standards help maintain consistency and quality throughout the development process, which makes the code easier to read, understand, and maintain. Coding Standards often cover naming conventions, indentation styles, commenting practices, file organization and error handling strategies. By using these guidelines, all the team members can write code in a same way, regardless of their personal styles which makes collaboration easier and efficient.

Importance of Coding Standards

Using coding standards is important for both teams and individuals, for individual developers they help to maintain a sense of discipline, making the code easier to debug and reduces the chances of errors caused by unclear logic and inconsistent styles. Whereas for teams, coding standards are essential to ensure different parts of the project developed by many people can be easily integrated. When multiple programmers are working on the same project, consistent code makes the code easier to read and lets any programmer understand and modify code written by others, this is very useful in projects like the KickBlast Judo program, where features such as BMI calculations, user input validation and GUI components are developed in different stages and need to work together smoothly.

Common Coding Standards

Some of the most common coding standards in modern programming include:

Consistent naming conventions: Classes, variables and methods should all be named in the same manner, for example, C# variables should be named in camelCase and class names should be named in PascalCase, this makes it easy to identify the purpose of each element

Proper indention and spacing: Indenting code blocks evidently help to keep the program structure clear, makes it easier to follow loops, conditions and nested elements, therefore makes it easier for the programmers as well

Meaningful comments: adding clear, understandable comments at important sections of the code, helps to explain the logic, makes future debugging and modification easier

Error handling practices: the use of try-catch blocks and meaningful error messages improves the stability of program and user experience, as these messages allow programmers to identify and remove errors in the system

Consistent file structure: organizing and maintaining code files logically, and dividing the responsibilities improves the quality of the system especially as it grows

These are coding standards commonly used in environments like Visual Studio when developing structured C# programs.

Applied Coding Standards in the KickBlast Judo Application with Explanations and Evidence

In the development of the KickBlast Judo System, many coding standards were consistently applied to maintain clarity, structure, and quality across the project. These standards ensured that the code was readable, easy to debug and simple to maintain throughout the development process.

1. Consistent Naming Conventions

Variables, classes and methods were named using camelCase and PascalCase appropriately. For example, variables such as totalFee, athleteName, and class names such as LoginForm and PaymentModule follow consistent naming rules, this makes it easy to identify the purpose of each element and avoid confusion when working across different forms of the system.

```
private void btnLogin_Click(object sender, EventArgs e)
{
    // Step 1: Validate inputs
    if (!ValidateInputs())
        return;

    // Step 2: Authenticate user
    string username = txtUsername.Text.Trim();
    string password = txtPassword.Text;

    if (AuthenticateUser(username, password))
    {
```

Figure 64: Proper naming conventions (Author developed, 2025)

This code shows proper consistent naming conventions, using camelCase for variables and PascalCase for class names, making the code clear and easy to maintain across the system.

2. Proper Indentation and Formatting

Throughout the entire system, proper indentation and spacing were used to clearly separate code blocks, conditions and loops, this helps to visually understand the program flow especially in areas with nested conditions or event handling.

```
// Method 12: Compare current weight with category limit
1 reference
private string CompareWeightWithCategory(double currentWeight, double categoryLimit)
{
    if (categoryLimit > 100) // Heavyweight
    {
        if (currentWeight > 100)
            return "Athlete is in correct weight category.";
        else
            return "Athlete can increase weight to match category.";
    }
    else
    {
        if (currentWeight <= categoryLimit)
            return "Athlete is in correct weight category.";
        else
            return "Athlete needs to reduce weight.";
    }
}
```

Figure 65: Example of properly indented if-else statements (Author developed, 2025)

3. Meaningful Comments

Inline comments and block comments were added to describe the logic of critical sections such as database connectivity, login validation, or fee calculation. This helps future

debugging and modifications process easier, especially in a system where there are multiple forms.

```
// Step 6: Open Main Dashboard (TEMPORARILY COMMENTED - Will enable after creating dashboard)

this.Hide();
frmMainDashboard dashboard = new frmMainDashboard();
dashboard.ShowDialog();

// Step 7: When dashboard closes, show login again
this.Show();
ClearFields();
LoggedInUser = "";


// TEMPORARY - Just clear fields for testing
ClearFields();
MessageBox.Show("Dashboard will open here after we create it!",
    "Info", MessageBoxButtons.OK, MessageBoxIcon.Information);
```

Figure 66: Example of meaningful comments (Author developed, 2025)

In the figure above, meaningful comments are given to show temporary code and future implementation steps for example, creating the dashboard, which helps keep the GUIs organized and comments clarify the purpose of each section code.

4. Error Handling Practices

Try–catch blocks were used in critical areas like database operations to handle exceptions, this improves stability and gives meaningful error messages, rather than crashing the KickBlast Judo application.

```
// Method 3: Load weight categories from database
1 reference
private void LoadWeightCategories()
{
    try
    {
        string query = "SELECT CategoryID, CategoryName FROM WeightCategories WHERE IsActive = 1 ORDER BY CategoryID";
        DataTable dt = DatabaseHelper.ExecuteQuery(query);

        cmbWeightCategory.DataSource = dt;
        cmbWeightCategory.DisplayMember = "CategoryName";
        cmbWeightCategory.ValueMember = "CategoryID";
        cmbWeightCategory.SelectedIndex = -1;
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Error loading weight categories: {ex.Message}",
            "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

Figure 67: Implementation of try–catch blocks to handle database-related errors in the system (Author developed, 2025)

5. Consistent File Structure

The KickBlast Judo project structure was organized logically, separating the forms, classes and modules to maintain clarity as the system expanded. Login forms, Main dashboard, Athlete registrations, Fee calculator were stored in relevant files and namespaces.

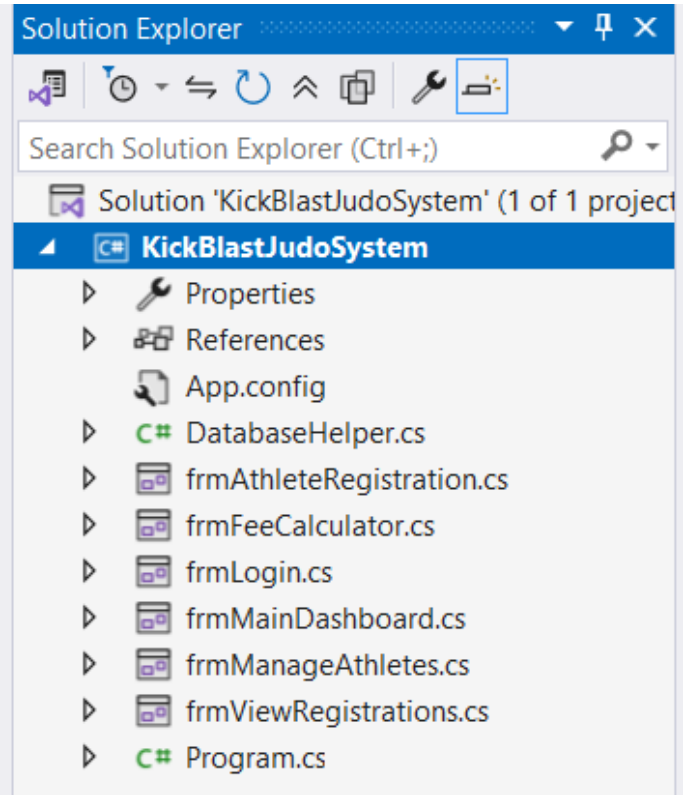


Figure 68: Organized file structure in Visual Studio for the KickBlast Judo System (Author developed, 2025)

Critical Evaluation on Why Coding Standards Are Essential for Teams and Individuals

In a professional software development environment such as CUBE-GEN Software Solutions, following clear and consistent coding standards is very important to supports the entire development lifecycle. As a Junior Software Developer working on the KickBlast Judo Application, I realized that coding standards was essential for ensuring that my code was understandable, maintainable and easy to integrate with a wider team structure.

When developing the GUI-based fee calculation system for KickBlast Judo, I was responsible for building various forms, implementing calculations, connecting to the SQL database and handling user input. Throughout this process, I applied the coding standards set by CUBE-GEN, including consistent naming conventions like camelCase for variables, PascalCase for classes, meaningful comments and proper indentation, these practices helped me as an individual developer while also making the project easier to share, debug, and maintain within a team environment.

Importance of Coding Standards for Team Development

In a team there will be multiple developers often working on different components of the same application simultaneously. Without shared coding standards, each developer might use their own style, which may result in inconsistency, confusion, and can be hard to maintain the code. For example, if one developer uses unclear variable names while another follows proper conventions, understanding and integrating the code becomes very difficult.

For the KickBlast Judo project, even though I developed most parts of the system by myself, I followed team-level standards so that any future developer at CUBE-GEN could understand and extend the application I created. This included applying consistent naming conventions for all forms, controls, variables, and methods. For instance, naming textboxes as txtAthleteName or txtTrainingPlan and form classes like LoginForm and ManageAthletesForm. This consistency means that if another developer needs to add any new features such as automated competition scheduling or reporting, they can quickly navigate the code without confusion.

Additionally, consistent commenting and documentation are essential in a team as well. In my application, I added comments above key methods and inside complex calculations. Like before calculating total fees, I added a short explanation describing the calculation logic, this makes it easier for another developer to understand my thought process without having to decipher the entire algorithm line by line.

Moreover, using proper indentation and logical structuring allowed me and my peers to use Visual Studio's debugging tools such as breakpoints, Step Into, Step Over, and Quick Watch more effectively. When debugging the fee calculation form, it was easy to identify logical sections of the code because methods were clearly and consistently named. In a real team environment, this would mean that multiple developers could debug and fix different modules at the same time without interfering with each other's work.

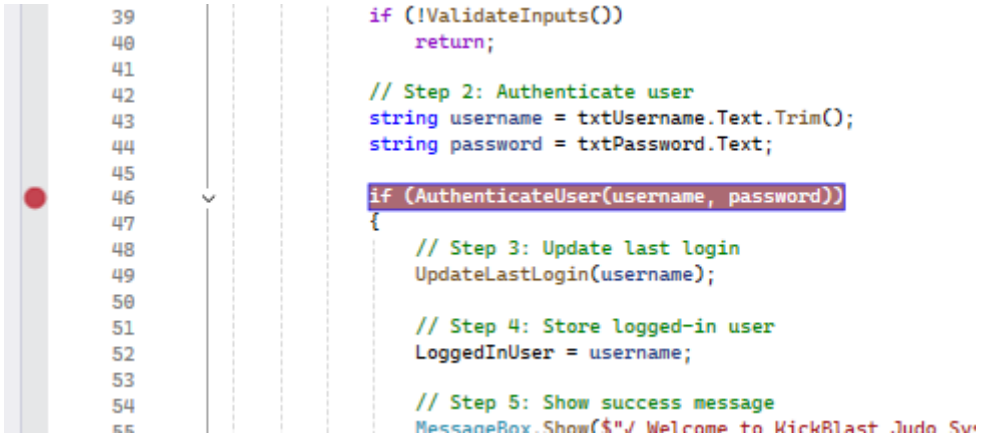


Figure 69: Breakpoint set at the start of the login method (Author developed, 2025)

Importance of Coding Standards for Individual Developers

As an individual developer, coding standards helped me to build discipline, reduce errors and improve overall productivity. When working on the KickBlast Judo application, having a clear structure for naming, formatting, and organizing code allowed me to think more clearly and focus on problem solving rather than wasting time remembering random naming choices or fixing inconsistent code.

For instance, by following camelCase for variable names like totalFee, athleteName, and PascalCase for classes like PaymentModule, LoginForm, I was able to quickly find whether

something was a method, class, or variable just by looking at it, this reduced confusion when switching between forms and modules.

Readability is another key benefit of Coding standards. Even when working alone, it is easy to forget why a piece of code was written in a certain way if you revisit it weeks later. By applying standards such as proper indentation, modular methods and inline comments, I was able to understand my own code during debugging and testing without wasting time deciphering it. For example, when testing the fee calculator, I was able to quickly find and fix a bug in the private coaching calculation because the code was neatly structured and commented.

```
// Method 9: Calculate private coaching cost (max 20 hours/month)
1 reference
private double CalculatePrivateCoachingCost(ref double hours)
{
    const double MAX_HOURS = 20.0;
    const double HOURLY_RATE = 90.50;

    if (hours > MAX_HOURS)
    {
        MessageBox.Show($"⚠ Maximum private coaching is 5 hours/week ({MAX_HOURS} hours/month).\n\n" +
            $"Hours adjusted to {MAX_HOURS}.",
            "Information", MessageBoxButtons.OK, MessageBoxIcon.Information);
        hours = MAX_HOURS;
        txtPrivateCoaching.Text = MAX_HOURS.ToString();
    }

    return hours * HOURLY_RATE;
}
```

Figure 70: Consistent naming conventions and meaningful comments applied in Fee Calculator module (Author developed, 2025)

Coding standards also helped me avoid common mistakes, clear naming conventions made it easier to avoid variable naming conflicts, while consistent indentation prevented me from missing misplaced brackets or nested logic errors. These small but important details reduced the number of runtime and logical errors I encountered during the development process.

From a career perspective, learning how to follow coding standards helps to build early professional habits. As a junior developer at CUBE-GEN, applying these standards consistently helps me to gain trust from senior team members and write code that meets the industry's expectations, this not only improves team collaboration but also enhances my personal growth and employability.

Critical Evaluation

Although it is possible to build small programs without strict coding standards, this approach is not suitable when multiple modules or developers are involved, the lack of structure leads to miscommunication, issues, and slower debugging. On the other hand, strictly used standards provide a shared language that allows both teams and individuals to work more efficiently.

For the KickBlast Judo project, these standards ensured that the application was not only functional but also professional, making it easy to hand over to another developer at CUBE-GEN in the future. As an individual, following these standards helped me develop strong coding discipline, reduced the number of errors and made my code more professional and understandable. In the real-world, maintaining this level of consistency would allow CUBE-GEN to scale projects efficiently, bring new developers into the team smoothly, and reduce the cost and time of maintenance. It also provides a collaborative environment where code is seen as a shared asset rather than personal work, which is important in professional software development.

References list

GeeksforGeeks, 2023. What is an Algorithm? Introduction, Types, Characteristics, and Examples. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/introduction-to-algorithms/ [Accessed 1 September 2025].

TutorialsPoint, 2023. Algorithms - Characteristics. [online] TutorialsPoint. Available at: https://www.tutorialspoint.com/design_and_analysis_of_algorithms/design_and_analysis_of_algorithms_characteristics.htm [Accessed 1 September 2025].

Encyclopedia Britannica, 2024. Fibonacci sequence. [online] Encyclopedia Britannica. Available at: https://www.britannica.com/science/Fibonacci-sequence [Accessed 1 September 2025].

TutorialsPoint, 2023. Factorial of a Number. [online] TutorialsPoint. Available at: https://www.tutorialspoint.com/factorial-of-a-number-in-cplusplus [Accessed 1 September 2025].

Programiz, 2024. Big-O Notation. [online] Programiz. Available at: https://www.programiz.com/dsa/asymptotic-notations#big-o-notation [Accessed 2 September 2025].

GeeksforGeeks, 2023. Programming Paradigms in Computer Science. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/programming-paradigms-in-computer-science/ [Accessed 2 September 2025].

TutorialsPoint (2023). Functions in Python. [online] TutorialsPoint. Available at: https://www.tutorialspoint.com/python/python_functions.htm [Accessed 19 September 2025].

Educative, 2021. Procedural programming vs object-oriented programming: What is the difference? [online] Educative. Available at: https://www.educative.io/blog/procedural-programming-vs-oop [Accessed 2 September 2025].

Rishitjain (2012) Syntax Error [online image]. Available at: https://rishitjain.wordpress.com/2012/05/17/python-elseelif-syntax-error-solved-and-indentation-concept/ [Accessed: 10 October 2025].

Tarry Singh (2024) The Big O notation and its significance [online image]. Available at: https://www.linkedin.com/pulse/big-o-notation-its-significance-llms-tarry-singh-vizxc/ [Accessed: 10 October 2025].

Jigar Shah (2023) Examples of most commonly used IDEs [online image]. Available at: https://radixweb.com/blog/what-is-ide [Accessed: 10 October 2025].

Jon Fincher (n.d.) Python Source code editor [online image]. Available at: https://realpython.com/python-development-visual-studio-code/ [Accessed: 10 October 2025].

IBM, 2023. What is an IDE? [online] IBM Developer. Available at: https://developer.ibm.com/articles/what-is-an-ide/ [Accessed 13 October 2025].

GeeksforGeeks, 2023. Debugger in IDE. [online image] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/debugger-in-ide/ [Accessed 13 October 2025].

Coding Ninjas, 2023. Coding Standards in Software Engineering. [online] Coding Ninjas. Available at: https://www.codingninjas.com/studio/library/coding-standards-in-software-engineering [Accessed 13 October 2025].

Pawal Grzybek, 2021. Step Over, Step Into and Step Out. [online image] Coding Ninjas. Available at: https://pawelgrzybek.com/continue-step-over-step-into-and-step-out-actions-in-visual-studio-code-debugger-explained/ [Accessed 16 October 2025].

Techopedia. (2024). What is Debugging? [online] Available at: https://www.techopedia.com/definition/16462/debugging [Accessed 16 Oct. 2025].

Microsoft, 2024. Code with IntelliSense in Visual Studio. [online] Microsoft Learn. Available at: https://learn.microsoft.com/en-us/visualstudio/ide/using-intellisense [Accessed 23 October 2025].